UNIVERSITY OF SOUTHAMPTON

Faculty of Engineering and Physical Sciences

EPSRC Centre for Doctoral Training in Next Generation Computational Modelling

Uncertainty quantification and propagation through complex chains of computational models

by

Stephen Gow

Thesis submitted for the degree of Doctor of Philosophy

January 2021

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

EPSRC Centre for Doctoral Training in Next Generation Computational Modelling

Doctor of Philosophy

Uncertainty quantification and propagation through complex chains of computational models

by Stephen Gow

There are many fields in which it is of interest to make predictions from a chain of computational models or simulators, in which the output of one simulator in the chain forms one of the inputs to the next simulator. In order to make reliable predictions from the chain, it is necessary to understand how uncertainty in the individual models will propagate through the chain. Each simulator will often be computationally intensive, and for computational feasibility must be approximated; we use a Gaussian process emulator to do this. This thesis focuses on a “linked” emulator, in which each model is emulated separately and the emulators are linked to make predictions from the chain as a whole.

We present two methods to make predictions from a chain of linked emulators. Both have precedent in previous research, but are fully formalised and extended in our work. One method uses simulation and Monte Carlo integration to make empirical predictions from the chain; this is extremely flexible and can be applied to a wide class of emulators, but can be computationally intensive and is open to Monte Carlo error. The second method uses theoretical results for the mean and variance of the linked emulator under certain restrictive conditions on the emulators of the individual models in the chain; this is fast and provides exact or near-exact results, but is possible only for a very limited set of emulators.

Related problems include experimental design and sensitivity analysis for chains of models. We present an algorithm for single-stage design, and discuss approaches to sequential design strategies. We also propose methods for sensitivity analysis on the final model of a chain, and develop techniques towards sensitivity analysis for the chain as a whole.

The above methodology is demonstrated on a chain to assess the impact of a chemical, biological or radiological release which combines a model for atmospheric dispersion with a model for the probability of casualty.

Contents

List of Figures

List of Tables

Declaration of Authorship

I, Stephen Gow, declare that the thesis entitled "Uncertainty quantification and propagation through complex chains of computational models" and the work presented in the thesis are both my own, and have been generated by me as the result of my own original research. I confirm that:

- this work was done wholly or mainly while in candidature for a research degree at this University;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- where I have consulted the published work of others, this is always clearly attributed;
- where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- none of this work has been published before submission.

Signed:

Date:

Acknowledgements

First and most importantly, I wish to thank my main supervisor, Professor David Woods, for his advice and guidance during the course of this PhD. Completing this thesis would not have been possible without the time and effort he provided to point me in the right direction, which must have required an extraordinary amount of patience at times. Thank you also to my secondary supervisor Jon Forster for his support, and to all those in the Mathematics department and in particular Southampton's design group who provided much-needed help.

I am also grateful to the team behind the Centre for Doctoral Training in Next Generation Computational Modelling (NGCM) - in particular Hans Fangohr, Ian Hawke and Susanne Ufermann - whose help during the taught year of the PhD and beyond made the coding side of my research much less daunting.

I must also acknowledge the Engineering and Physical Science Research Council (EPSRC) and the Defence Science and Technology Laboratory (Dstl) for the funding contributions that made this project possible. Special thanks to Veronica Bowman and Paul Westoby of Dstl, who provided valuable advice and assistance with the real-life applications of our work.

Finally, I would like to thank my family and friends for their support over the years.

Glossary of symbols

Symbol	**Meaning**
y	Output of a single-output computational model
q	Number of inputs to a computational model
x_q	Input q to a computational model
$\mathbf{x} = (x_1, x_2, ..., x_q)^T$	Vector of inputs to computational model
$\eta(\mathbf{x})$	Model function (simulator) for computational model
$\mathbb{R}^q$	q-dimensional real number space
$\mathcal{X}$	Space on which the model inputs are defined (subset of $\mathbb{R}^q$)
n	Number of design points for a computer experiment
$\xi = (\mathbf{x}_1, ..., \mathbf{x}_n)^T$	Experimental design with n design points
$y_1, ..., y_n$	Outputs of a computer experiment with n design points
$Z(\mathbf{x})$	Gaussian process
k	Dimension of any subset of $\mathcal{X}$ in GP definition
$\mu_z(\mathbf{x})$	Mean function of a Gaussian process
$C_z(\mathbf{x}_1, \mathbf{x}_2)$	Covariance function of a Gaussian process
$\mathbf{h} = \mathbf{x}_1 - \mathbf{x}_2$	Vector of distance between two points
$C(\mathbf{h})$	Stationary covariance function of a GP
σ_z^2	(Constant) process variance of a GP
$R(\mathbf{h}) = C(\mathbf{h})/\sigma_z^2$	Correlation function of a GP
b_j	Scale parameter of power-exponential correlation function in dimension j
α	Shape parameter of power-exponential correlation function
θ_j	Scale parameter of Matérn correlation function in dimension j
ω	Smoothness parameter of Matérn correlation function
p	Number of regression terms in GP emulator
$\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), ..., f_p(\mathbf{x})]$	Vector of regression functions for GP emulator
$\boldsymbol{\beta} = (\beta_1, ..., \beta_p)^T$	Vector of regression coefficients for GP emulator
$\boldsymbol{\mu}$	Mean of ordinary GP emulator
$\mathbf{x}_{n+1}$	New input vector at which we wish to predict from GP emulator
Y_{n+1}	Unknown simulator output at new input vector
$\mathbf{Y_n}$	Vector of results of a computer experiment

$\mathbf{F}$	Matrix of regression functions for the elements of ξ
$\mathbf{f}_{n+1}$	Vector of regression functions for $\mathbf{x}_{n+1}$
$\mathbf{C}$	Matrix of correlations among the elements of ξ
$\mathbf{c}_{n+1}$	Vector of correlations between $\mathbf{x}_{n+1}$ and ξ
$\boldsymbol{\theta} = (\theta_1, ..., \theta_q)^T$	Vector of correlation parameters
$\mathbf{b}_0$	Mean of normal prior on $\boldsymbol{\beta}\|\sigma_z^2$ or $\boldsymbol{\beta}$
$\mathbf{V}_0$	Variance matrix of normal prior on $\boldsymbol{\beta}\|\sigma_z^2$ or $\boldsymbol{\beta}$
ν_0	Degrees of freedom of scaled-inverse-chi-squared prior on σ_z^2
c_0	Parameter used in definition of scaled-inverse-chi-squared prior on σ_z^2
T_1	Univariate non-standardised t-distribution
ν	Degrees of freedom of non-standardised t-distribution
$\boldsymbol{\mu}$	Location parameter of multivariate non-standardised t-distribution
$\boldsymbol{\Sigma}^2$	Scale parameter matrix of multivariate non-standardised t-distribution
ν^*	Scale parameter of non-standardised t-distribution for prediction from GP emulator
μ^*	Location parameter of normal or non-standardised t-distribution for prediction from GP emulator
σ^{*2}	Scale parameter of normal or non-standardised t-distribution for prediction from GP emulator
$\hat{\boldsymbol{\beta}}$	Posterior estimate for an unknown $\boldsymbol{\beta}$
τ^2	Parameter used in definition of Normal prior on $\boldsymbol{\beta}$
δ	Nugget of a GP emulator
I	Identity matrix
$\hat{\sigma}_z^2$	Maximum likelihood estimator for σ_z^2
S	State space of a Markov chain
A	Subset of S
$\boldsymbol{\theta}^{(j)}$	Value of $\boldsymbol{\theta}$ in a Markov chain after j timesteps have elapsed
ψ	State of a Markov chain
ϕ	State of a Markov chain not equal to ψ
π	Stationary distribution of a Markov chain
M	Monte Carlo sample size
$t(\boldsymbol{\theta})$	Function of $\boldsymbol{\theta}$ we wish to approximate using Markov chain Monte Carlo sample
t'_M	Approximation to $t(\boldsymbol{\theta})$ of size M
$D(\boldsymbol{\theta})$	Target distribution of a Markov chain Monte Carlo simulation
$q(\psi)$	Transition kernel in a Markov chain Monte Carlo simulation
$\mathbf{w}_j$	Movement term in a random walk transition kernel
f_w	Probability density function of $\mathbf{w}_j$

$f_D(\boldsymbol{\theta})$	Function proportional to target density $D(\boldsymbol{\theta})$
$\boldsymbol{\Theta}$	Sample matrix from $D(\boldsymbol{\theta})$ obtained using Markov chain Monte Carlo
λ_i	ith eigenvalue of a matrix Λ
y_k	Output of model k in a chain of models
q_k	Number of inputs to model k in a chain, excluding those which depend on a previous model
$\tilde{\mathbf{x}}_k = (x_{k,1}, ..., x_{k,qk})^T$	Vector of inputs to model k in a chain, excluding inputs which depend on a previous model
$\eta_k(\tilde{\mathbf{x}}_k, y_{k-1})$	Model function (simulator) for model k in a chain
$\boldsymbol{\beta}_k$	Vector of regression coefficients for GP emulator of model k in a chain
$\sigma^2_{z,k}$	Process variance of GP emulator of model k in a chain
$\boldsymbol{\theta}_k$	Vector of correlation parameters for GP emulator of model k in a chain
$\mathbf{F}_k$	Matrix of regression functions at the design points for model k in a chain
$\mathbf{C}_k$	Matrix of correlations among the design points for model k in a chain
$\mathbf{Y}_{n,k}$	Vector of results of a computer experiment for model k in a chain
$\tilde{\mathbf{x}}_{k,n+1}$	Vector of directly controllable inputs to model k at an untested input configuration
$\mathbf{x}_{k,n+1}$	Vector of inputs to model k in a chain at an untested input configuration
$\mathbf{f}_{n+1,k}$	Vector of regression functions for $\mathbf{x}_{k,n+1}$
$\mathbf{c}_{n+1,k}$	Vector of correlations between $\mathbf{x}_{k,n+1}$ and the design points for model k
$f^*(y_2)$	Product of conditional densities for y_1 and y_2 in a two-model linked emulator
$d(\mathbf{x}_i, \mathbf{x}_j)$	Distance between the points $\mathbf{x}_i$ and $\mathbf{x}_j$ in χ
$\phi_{Mm}(\xi)$	Maximin distance design criterion
$\phi_{mM}(\xi)$	Minimax distance design criterion
$C_{k,l}(\xi)$	Coverage design criterion for parameters k and l
S	Set of candidate points in χ
n_k	Number of design points in computer experiment for model k in a chain
ξ_k	Experimental design for model k in a chain
$\mathbf{L}_k$	Matrix of limits for controllable inputs to model k in a chain
$\mathbf{L}_{y,k}$	Vector of length 2 containing limits for y_{k-1} based on simulator runs at points in ξ_{k-1}

y_1^{new}	Value of y_1 at a newly added design point in $\tilde{\mathbf{x}}_1$ in sequential experimental design
κ	Set of indices in q
$\mathbf{x}_\kappa$	Subvector of $\mathbf{x}$ containing the elements with indices in κ
$\mathbf{x}_{-\kappa}$	Subvector of $\mathbf{x}$ containing elements with indices not in κ
$G(\mathbf{x})$	Probability distribution for $\mathbf{x}$
$E(Y\|\mathbf{x}_\kappa)$	Expectation of model output when the inputs in the subvector $\mathbf{x}_\kappa$ are fixed
$z_i(x_i)$	Main effect of input x_i
$z_{i,j}(\mathbf{x}_{i,j})$	Second-order interaction between the inputs x_i and x_j
V_κ	Expected reduction in the variance of Y when $\mathbf{x}_\kappa$ is fixed
S_κ	Sobol' index for subset of inputs $\mathbf{x}_\kappa$
$V_{T\kappa}$	Expected uncertainty when all inputs other than $\mathbf{x}_\kappa$ are fixed
$S_{T\kappa}$	Total effect index for subset of inputs $\mathbf{x}_\kappa$
$G_\kappa(\mathbf{x}_\kappa)$	Marginal distribution of $\mathbf{x}_\kappa$
$G_{-\kappa\|\kappa}(\mathbf{x}_{-\kappa}\|\mathbf{x}_\kappa)$	Conditional distribution of $\mathbf{x}_{-\kappa}$ given $\mathbf{x}_\kappa$
EE_i	Elementary effect of input x_i
ρ	Number of levels in the discretisation of the input space for elementary effect analysis
Δ	Value in the set $\left[\frac{1}{\rho-1},\frac{2}{\rho-1},...,1-\frac{1}{\rho-1}\right]$ used in elementary effects analysis
μ_i^*	Measure of the sensitivity of Y to the input x_i using elementary effect analysis
χ_κ	Design space of the inputs $\mathbf{x}_\kappa$
$\chi_{-\kappa}$	Design space of the inputs $\mathbf{x}_{-\kappa}$
$\mathbf{R}_\kappa(\mathbf{x}_\kappa)$	Integral of $\mathbf{f}_{n+1}$ with respect to $G_{-\kappa\|\kappa}(\mathbf{x}_{-\kappa})$ across $\chi_{-\kappa}$
$\mathbf{T}_\kappa(\mathbf{x}_\kappa)$	Integral of $\mathbf{c}_{n+1}$ with respect to $G_{-\kappa\|\kappa}(\mathbf{x}_{-\kappa})$ across $\chi_{-\kappa}$
$\mathbf{R}$	Special case of $\mathbf{R}_\kappa(\mathbf{x}_\kappa)$ where κ is the empty set
$\mathbf{T}$	Special case of $\mathbf{T}_\kappa(\mathbf{x}_\kappa)$ where κ is the empty set
$R^*(\mathbf{x},\mathbf{x}')$	Posterior correlation between $\mathbf{x}$ and $\mathbf{x}'$
υ	Set of indices in q which are not identical to κ
$U_{\kappa,\upsilon}(\mathbf{x}_\kappa,\mathbf{x}'_\upsilon)$	Integral of $R^*(\mathbf{x},\mathbf{x}')$ with respect to the conditional distributions of two distinct sets of unknown inputs
$\mathbf{E}$	Triple integral of the product $\mu^*(\mathbf{x})\mu^*(\mathbf{x}^*)$ with respect to the marginal distribution $G_\kappa(\mathbf{x}_\kappa)$ and the conditional distributions $G_{-\kappa\|\kappa}(\mathbf{x}_{-\kappa}\|\mathbf{x}_\kappa)$, $G_{-\kappa\|\kappa}(\mathbf{x}'_{-\kappa}\|\mathbf{x}'_\kappa)$
M	Monte Carlo sample size
$S_\kappa(\mathbf{x}_\kappa)$	Distribution used in importance sampling for $G_\kappa(\mathbf{x}_\kappa)$
$S_{-\kappa\|\kappa}(\mathbf{x}_{-\kappa}\|\mathbf{x}_\kappa)$	Distribution used in importance sampling for $G_{-\kappa\|\kappa}(\mathbf{x}_{-\kappa}\|\mathbf{x}_\kappa)$
$Cov^*(Y_2,Y_2')$	Covariance between two independent realisations of a two-model linked emulator given the directly controllable inputs

$c^*(Y_2, Y_2')$	Covariance between two independent realisations of a two-model linked emulator given all inputs

Chapter 1

Introduction

A computational models is a complex mathematical model implemented via a computer program to simulate a real-life process. Computational models are commonplace, being used in "almost all fields of science and technology" according to O'Hagan (2006). The method of constructing a mathematical model for a real process predates computers, and allows the modeller to gain insight into the behaviour of the process without the need for potentially difficult or expensive real-world testing. Implementing the model as a computer program allows more complex models to be considered than could be done by hand. In this thesis, we consider computational models which return a single output value, which we call y. A single-output model may nonetheless depend on multiple inputs, say q, which we group into a vector $\mathbf{x} = (x_1, x_2, ..., x_q)^T$. We work with models of the form

$$y = \eta(\mathbf{x})\,,$$

where $\eta(\mathbf{x})$ is a deterministic function. The model function $\eta(\mathbf{x})$ is the core of the computational model itself. Following the terminology of O'Hagan (2006), we call the model a simulator. This form of model does not include a random term; the output y is entirely determined by the inputs $\mathbf{x}$ and the function $\eta(\mathbf{x})$, so the same output is obtained any time the model is run with the same inputs. Uncertainty in the simulator output thus arises only from uncertainty in its input variables.

These restrictions on the type of model we work with are somewhat limiting, as a deterministic model with a single output will not always be an appropriate choice. For some real-life applications, computational models with multiple outputs are more useful than single-output models. These will be not be dealt with here. Additionally, it is highly unlikely that a computational model is exactly equivalent to the real-life process it is designed to replicate, so a deterministic model is not necessarily the best approximation. A model which attempts to quantify the uncertainty in its own output via a probability distribution may provide a better reflection of the real process being modelled. Despite this, deterministic models are widely used across many application areas, and form the basis of a large body of statistical literature on the topic of computational models.

When the behaviour of a process across the entire space of its inputs is of interest, the simulator must be run many times for different input configurations. This can be computationally infeasible if the simulator is expensive to run. Computer experimentation provides a solution to this: the simulator is run for a few input configurations, called training or design points, and the outputs are treated as data from an experiment, which is then used to construct a statistical approximation to the simulator. The approximation is called an emulator, and the technique - which was first applied to computational models by Sacks et al. (1989) - is called emulation. Predictions at other inputs can then be made quickly from the emulator. This approach does however introduce uncertainty into the predictions made; to quantify this, the emulator returns a probability distribution instead of a single value. Analysis of uncertainty in the computational model must thus take account of not only the uncertainty arising from the inputs, but also the uncertainty in the emulator at input configurations where the simulator has not been run directly. The most common emulator uses a combination of a regression model and a stationary Gaussian process to approximate the simulator. Emulation for a single model is discussed in Chapter 2.

The overall aim of this thesis is to explore how predictions can be made and assessed through chains of computational models, in which the output of one model in the chain is then used as an input (potentially one of several) to the next model until a final model is reached. The models may be extremely complex and computationally intensive, so emulation will be required. Each link in the chain is thus subject to two sources of uncertainty: uncertainty arising from the inputs to the model, and uncertainty arising from the emulator. These individual sources of uncertainty will propagate and combine through the model chain, affecting our ability to understand and quantify the reliability and accuracy of overall predictions from the full chain.

Motivation for the project comes from the multi-modal chains of computational models used by our research partners, the Defence Science and Technology Laboratory (Dstl), for hazard prediction and management. For example, consider the prediction of casualties from a chemical, biological or radiological (CBR) release. An illustration of this process can be seen in Figure 1.1. One output of interest is the probability of casualty for an affected individual. This output depends on several linked processes which would each need to be approximated by separate models which would then be linked together. The probability of casualty is a function of the properties of the released contaminant and the dosage received by the individual; this would constitute final model in the chain. The dosage received is itself a function of several other variables determining how the contaminant reaches the individual following its release. These variables could include the location of the release, the release mass and duration, and the wind speed and direction both at and after the time of release. At least one further model would be needed to capture this variation, and potentially another to account for changes in meteorological conditions across time. For a fuller picture of the effects of such a release on the population as a whole, the probability of casualty at each location in the potentially affected area would itself be an input into a final model for the number of casualties due to the release. This model could also include other variables such as the population density at each location, placement of sensors to detect the release and factors relating to the strategy chosen to mitigate the effects of the release, allowing the effects of different mitigation strategies to be modelled.

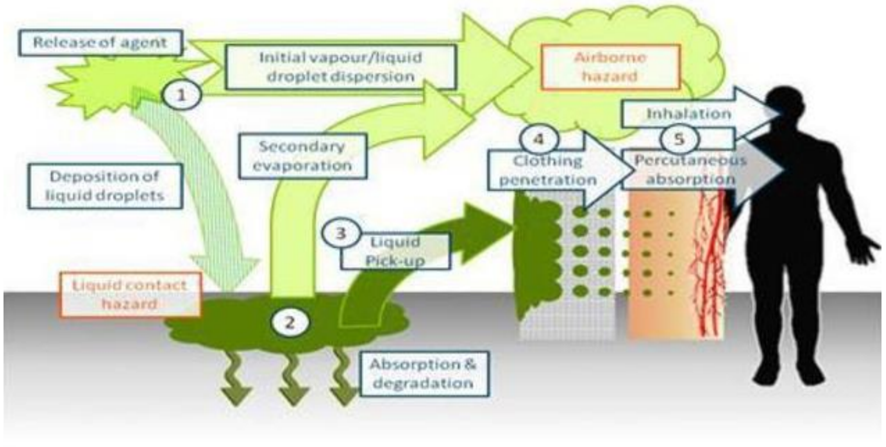

Figure 1.1: Diagram of the steps in a chain of models to predict the casualties from a CBR release. (Source: Dstl)

There are two alternative methods by which emulation for chains of models can be conducted. The first is to approximate the entire chain via a single emulator, referred to in previous work on the topic as the composite emulator. This has the advantage of in effect reducing the problem to one which has already been solved, as the theory for single-model emulation is extremely well-developed. However, if the input space of the chain of models is large, this approach may be computationally expensive; knowledge of the chain of models effectively defines a form of dimension reduction on the full chain. Additionally, information can be lost by performing only one emulation, both in terms of understanding the behaviour of subsets of the models in the chain and - as in an example in the work of Kyzyurova et al. (2018) - in terms of the performance of the composite emulator.

The second approach, which is taken during our work, is to emulate each model independently and link the results together to produce a final approximation called the linked emulator. This introduces additional challenges, since one of the inputs to the later models in the chain is not known exactly but only up to a probability distribution. Approaches exist to deal with this uncertainty for a chain of two models using either Monte Carlo methods (Kyzyurova et al., 2018) or theoretical results concerning inputs to a Gaussian process emulator which follow a normal distribution (Candela et al., 2003; Kyzyurova et al., 2018). Chapter 3 reviews the existing methods and extends them to chains of more than two models.

There is a relationship between this approach and the field of deep Gaussian processes. A deep Gaussian process is a form of neural network based on a Gaussian process prior distribution, with its origins in the work of Neal (1996), Chapter 2, which focuses on priors for Bayesian neural network. More recently, Damianou and Lawrence (2013) developed a framework for deep learning in which the observed data is treated as the output of a multivariate Gaussian process, and the inputs to this Gaussian process are themselves controlled by another Gaussian process. This can be thought of a form of latent variable modelling in which the Gaussian process controlling the inputs to the second Gaussian process adds an additional layer of understanding to the behaviour of the system as a whole. Damianou and Lawrence (2013) go on to use variational marginalisation to remove all of the intermediate layers from the system, with some level of approximation required to do this.

The structure of this hierarchy can be viewed as similar to that of a chain of models. There are however some noteworthy differences between the deep Gaussian process setup and the chain of models as defined in this thesis. Firstly, each model in a chain takes only one input as an output from a previous model, instead of an entire layer at a time being defined through latent variables. More importantly, every input and output to any model in the chain corresponds directly to a quantity of interest, and their definition comes directly from the real-world process being modelled. The number of intermediate inputs is fixed by the nature of the problem. In a deep Gaussian process, the latent variables are typically artificial, and the number of intermediate inputs can be chosen for the purposes of improving predictive performance. Our work is nonetheless similar in spirit to the method of Damianou and Lawrence (2013), but differs significantly in its approach.

A further problem of interest is that of how the design points for the chain of computer experiments required to build each emulator should be chosen. There is a large body of theory for how this can be done for a single model. One popular approach is space-filling design, in which the points are chosen to fill the input space based on either a Latin hypercube principle (McKay et al., 1979), or to satisfy an optimality criterion based on the distances between the points (Johnson et al., 1990). A second method is sequential design, in which the model is run for a reduced set of design points and the remaining points are allocated using information gained from the runs which have already been made (Sacks et al., 1989; Gramacy and Lee, 2009). Extending either of these methods to models in a chain faces challenges not seen in the single-model case. Both the existing methods and our proposals for experimental design for chains of models are discussed in Chapter 4.

The quantification and understanding of uncertainty in statistical models forms the basis of the field of sensitivity analysis. When applied to a deterministic model, sensitivity analysis is based upon apportioning the uncertainty in the simulator output to its inputs (Saltelli et al., 2008). Several techniques exist to do this; the approaches we are most interested in are based on decomposing the model into a sum of the main effects of each input and the interactions between them, and on quantifying the effect of each input (or set of inputs) in terms of the proportion of the total output variance explained by the input(s). However, traditional sensitivity analysis relies on an extremely large number of model runs at different input configurations, so it is often necessary to use emulation to build an approximation to the simulator for sensitivity analysis to proceed (Oakley and O'Hagan, 2004). Sensitivity analysis for a chain of emulators is thus of interest. Chapter 5 focuses on both existing methods for sensitivity analysis for a single model, including an example from Dstl based on CBR modelling, and their potential extensions to the case of a chain.

The methods developed in the early chapters of the report are implemented in the `R` programming language, with some use of `C++` for reasons of execution speed. Our code includes functions for prediction from linked emulators using both the theoretical and the Monte Carlo method, methods for sensitivity analysis for the final model of a chain given its own inputs, and some limited sensitivity analysis for the output of the chain with respect to the directly controllable inputs. This software implementation is introduced in Chapter 6.

The techniques we present for prediction and analysis of chains of computational models are demonstrated on a real-life example from Dstl in Chapter 7. The study concerns a chain of two models for dispersion and casualty estimation from a CBR release. We construct a linked emulator for the chain using both the Monte Carlo and the theoretical method and review the performance of both, together with that of a composite emulator. Sensitivity analysis for the CBR chain is also considered.

It is of interest to make comparisons between the different methods considered in this thesis. In Chapter 8, a simulation study is conducted to investigate the differences in behaviour between the two forms of linked emulator and the simpler methods of a composite emulator and linear regression, and to determine which settings the different approaches are best suited to. Finally, the main conclusions of our work and ideas for relevant future research directions are presented in Chapter 9.

Chapter 2

Emulation

2.1 Overview

Emulation is a technique which attempts to reduce the time and cost associated with prediction from a computationally expensive simulator. In theory, the output of the simulator can be determined for any choice of inputs we wish to make; if, as we assume to be the case throughout, the model is deterministic, this output does not change should we run the simulator multiple times with the same inputs. In practice, for many real-world problems the simulator is computationally expensive or time-consuming to run, and it is therefore infeasible to perform a large number of runs of the simulator. But there are several situations in which a large number of simulator runs would be required. It may be necessary, for example, to make predictions at a large number of input configurations relating to different real-world conditions. Additionally, sensitivity analysis for a computational model can require many thousands or even millions of model runs; this shall be discussed further in Chapter 5.

When using the simulator directly is infeasible, the most practical alternative is to construct an approximation to the simulator using a computer experiment. Let $\mathcal{X} \subset \mathbb{R}^q$ be the space on which the q inputs to the simulator are defined; in the context of a computer experiment, $\mathcal{X}$ is called the design space. The simulator is run at a relatively small number of input settings, $\xi = (\mathbf{x}_1, ..., \mathbf{x}_n)^T, \mathbf{x}_1, ..., \mathbf{x}_n \in \mathcal{X}$, called design points or training points. The (scalar) outputs, $y_1, ..., y_n$, are treated as data from an experiment. This data is then used to build a statistical approximation to the simulator, called an emulator, which can then be used to make predictions of the simulator output for untested input values. The key advantage of this method is to substantially reduce the computational cost associated with prediction from the original model.

The simulator itself is a function of its inputs $\mathbf{x}$ which returns a single value, $y = \eta(\mathbf{x})$. For a sufficiently good approximate function $\hat{\eta}$ of η, it would be possible to take the point approximation $\hat{y} = \hat{\eta}(\mathbf{x})$ as a surrogate for y and perform any desired analysis on this approximation. This approach is sometimes taken in previous literature: a recent example is the work of Joseph et al., 2019 on model-based optimal experimental design, where it is used due to its computational simplicity. However, this approach does not reflect the fact that we cannot be sure of the accuracy of a point approximation where the true simulator output is unknown. It is therefore preferable to quantify the uncertainty in the estimate. For this reason, the most common emulators return a probability distribution for the true output y given the inputs $\mathbf{x}$ instead of a single value. Bayesian methodology can be used to construct an emulator from the training data.

There are two properties which a good emulator typically obeys. At the design points, where the true output is known, we would expect the emulator to return the known output with probability one. Elsewhere, the probability distribution returned by the emulator should have a mean which is a plausible estimate for the true output, and should provide a reasonable expression of the uncertainty associated with this estimate. The first property can be easily checked, but the second is somewhat more difficult; one common approach, as discussed by Bastos and O'Hagan (2009), is to run the model at some previously-untested inputs and compare the true output at these points to the emulator output.

The most common form of emulator in use is the Gaussian process emulator, which we discuss in Section 2.2; it is based upon a stochastic process called the Gaussian process. This can be interpreted within a Bayesian framework: the simulator is treated as an unknown function, with a prior distribution given by a Gaussian process. When combined with the data obtained from the simulator runs, this leads to a posterior distribution for the simulator output at any point. The resulting posterior distribution is also Gaussian, with a mean and covariance which depend on both the data and the parameters of the prior distribution.

The choice of the points at which the true model is run, called the experimental design, is important: a well-chosen set of design points allow much more information to be gained from the simulator runs than a poorly-chosen set. Several methods exist to choose the points well for a single model. These are reviewed in Section 4.1 in the context of extending the process to multiple linked models.

2.2 Gaussian Process emulation

Using the notation followed by Santner et al. (2003), let $Z(\mathbf{x}), \mathbf{x} \in \mathcal{X}$, be a stochastic process. $Z(\mathbf{x})$ is a Gaussian process if $Z(\mathbf{x}_1), ..., Z(\mathbf{x}_k)$ follows a k-dimensional multivariate normal distribution for any subset $(\mathbf{x}_1, ..., \mathbf{x}_k)$ of $\mathcal{X}$. The alternative nomenclature Gaussian random function or Gaussian random field is used in some literature, but we will use the more common term Gaussian process (GP). A GP is defined by its mean function,

$$\mu_z(\mathbf{x}) = E[Z(\mathbf{x})] \, ,$$

for $\mathbf{x} \in \mathcal{X}$, and by its covariance function ,

$$C_z(\mathbf{x}_1, \mathbf{x}_2) = Cov[Z(\mathbf{x}_1), Z(\mathbf{x}_2)]\,,$$

for $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$. Both μ_z and C_z may be parametric functions with an additional dependences on unknown parameters. A covariance function is defined as stationary if, where $\mathbf{x_1} - \mathbf{x_2}$ is the distance between $\mathbf{x_1}$ and $\mathbf{x_2}$, the covariance is given by

$$Cov[Z(\mathbf{x}_1), Z(\mathbf{x}_2)] = C(\mathbf{x}_1 - \mathbf{x}_2)\,,$$

for some function C. For ease of notation, we write $\mathbf{h} = \mathbf{x}_1 - \mathbf{x}_2$, with h_j being the jth element of the vector $\mathbf{h}$. A stationary covariance function $C(\mathbf{h})$ thus depends on $\mathbf{h}$ and any other parameters of the function. GPs with stationary covariance functions have many useful properties regarding inference based on data from a single sample path; details are given by Adler (1981).

The process variance of a GP is defined as

$$\begin{aligned}\sigma_z^2(\mathbf{x}) &= Var[Z(\mathbf{x})] \\ &= Cov[Z(\mathbf{x}), Z(\mathbf{x})]\,.\end{aligned}$$

An additional assumption which is usually made is that the process variance is a constant value σ_z^2. This variance must be greater than zero for the GP to be non-degenerate. For a stationary GP, the process variance can be written as

$$\sigma_z^2 = C(\mathbf{0})\,.$$

It is often more convenient, and substantially more common in the literature, to work with the correlation function of a stationary GP instead of the covariance function. This is defined as:

$$R(\mathbf{h}) = C(\mathbf{h})/\sigma_z^2\,.$$

For the GP to be non-degenerate, the correlation function must satisfy $R(\mathbf{0}) = 1$; for stationarity, we additionally require that $R(\mathbf{h}) = R(-\mathbf{h})$, and that R is positive semi-definite. It is useful to note that the product of several correlation functions with these properties is itself a valid correlation function with the same properties. This is referred to as separability, and means that it is possible to create multi-dimensional correlation functions by taking the product of several independent one-dimensional correlation functions meeting the required conditions. This is useful, as for the purposes of building a GP emulator, the correlation function must be of dimension q, where q is the number of inputs to the computational model.

Santner et al. (2003), Chapter 2, presents several families of multi-dimensional correlation functions which meet the conditions required for non-degeneracy and stationarity, of which two are of particular interest to us. The power-exponential family of correlation functions in q dimensions has the form

$$R(\mathbf{h}) = \exp\left(- \sum_{j=1}^{q} b_j |h_j|^{\alpha} \right) . \tag{2.1}$$

This is a parametric function: each $b_j > 0$ is a scale parameter for the jth dimension, while α is a shape parameter which is consistent across dimensions, and must satisfy $0 < \alpha \leq 2$. The most commonly used value for this parameter is $\alpha = 2$, the Gaussian correlation function,

$$R(\mathbf{h}) = \exp\left(- \sum_{j=1}^{q} b_j h_j^2 \right) , \tag{2.2}$$

or equivalently

$$R(\mathbf{h}) = \prod_{j=1}^{q} \exp(-b_j h_j^2) \, .$$

This has been used in prior work on emulation and sensitivity analysis, notably by Oakley and O'Hagan (2004), and is the basis of previous work on linking Gaussian process emulators in a chain by Girard et al. (2002) and Kyzyurova et al. (2018). A Gaussian process with this correlation function is infinitely differentiable, and tends to produce extremely smooth sample realisations (Stein, 1999).

Another common choice is the Matérn correlation function, introduced in the PhD thesis of Matern (1960), which has the form

$$R(\mathbf{h}) = \prod_{j=1}^{q} \frac{1}{\Gamma(\omega) 2^{\omega - 1}} \left(\frac{2\sqrt{\omega}|h_j|}{\theta_j} \right)^{\omega} K_{\omega} \left(\frac{2\sqrt{\omega}|h_j|}{\theta_j)} \right) , \tag{2.3}$$

in q dimensions. Each $\theta_j > 0$ is the jth-dimensional scale parameter. $\omega > 0$ is a smoothness parameter; the smoothness parameters could also be dimension-specific, but authors such as Santner et al. (2003) use a common smoothness parameter in their definitions. K_{ω} is the modified Bessel function of the second kind of order ω. If ω is an integer multiple of $1/2$, the Bessel function reduces to a much simpler form. The most common choices for the smoothness parameter are $\omega = 5/2$ and $\omega = 3/2$. The alternative $\omega = 1/2$, which in one dimension leads to a GP which is equivalent to the Ornstein-Uhlenbeck process, is generally too rough for computer experiments; $\omega \geq 7/2$ gives very smooth realisations similar to that of the squared exponential, which is itself a limiting case of the Matérn correlation function as $\omega \to \infty$ per Rasmussen and Williams (2006), Chapter 4.

For the purposes of emulation, we could use a Gaussian process to approximate a simulator. However, more flexibility is often required than can be provided by a GP in isolation. This can be achieved by considering the output of the simulator as a realisation of a stochastic process, and fitting a regression model to this stochastic process; instead of modelling the error term as a series of independent and identically distributed random variables, it is modelled as a zero-mean stationary Gaussian process. This leads to the model

$$\begin{aligned} Y(\mathbf{x}) &= \sum_{i=1}^{p} f_i(\mathbf{x})\beta^{(i)} + Z(\mathbf{x}) \\ &= \mathbf{f}^T(\mathbf{x})\boldsymbol{\beta} + Z(\mathbf{x}) \, , \end{aligned}$$

where p is the the number of regression terms in the model, $\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), ..., f_p(\mathbf{x})]^T$ is a vector of regression functions, $\boldsymbol{\beta} = (\beta_1, ..., \beta_p)^T$ is a vector of regression coefficients and $Z(\mathbf{x})$ is a stationary Gaussian process with zero mean. Since Z has mean 0, its behaviour is determined entirely by the choice of the process variance and the correlation function. We shall refer to this technique as Gaussian process emulation; it is also called Kriging by authors including Fang et al. (2006), after a similar and longer-established technique from mining and geostatics. It was first applied to computer experiments by Sacks et al. (1989).

The choice of regression component of the Gaussian process emulator is an important one. Intuitively, it is tempting to choose a model with several terms, as a well-fitted regression model can explain much of the variation in the simulator output. In practice, however, it is rare to perform regression using an order higher than linear; this is justified by O'Hagan (2006), who states that practical experience suggests that the additional complexity of a higher-order model does not lead to a substantial enough improvement in the fit to be worthwhile. In the literature, the most common model consists of a constant regression term (the mean) plus the stationary Gaussian process,

$$Y(\mathbf{x}) = \boldsymbol{\mu} + Z(\mathbf{x}) \, .$$

This is equivalent to setting the number of regression terms $p = 1$, and is referred to by some authors, including Fang et al. (2006), as the ordinary Kriging model; the sample mean is used as a point estimate for $\boldsymbol{\mu}$. A common alternative is $p = 2$, a linear regression component, used for example by O'Hagan (2006).

To predict the simulator output, $Y_{n+1} = \eta(\mathbf{x}_{n+1})$, for an untested input vector, $\mathbf{x}_{n+1}$, given the known vector $\mathbf{Y_n} = [\eta(\mathbf{x}_1), ..., \eta(\mathbf{x}_n)]^T$ of the simulator output at the training points, we must use the posterior predictive distribution for Y_{n+1}. This is identical to the conditional distribution of the unknown simulator output given the training data, so we must derive the density $f(Y_{n+1}|\mathbf{Y}_n)$.

As before, let n be the number of design points for the emulator and p be the number of regression terms. We define $\mathbf{F}$ as an $n \times p$ matrix of regression functions for the elements of the experimental design ξ, $\mathbf{f}_{n+1} = \mathbf{f}(\mathbf{x}_{n+1})$ as a $p \times 1$ vector of regression functions for $\mathbf{x}_{n+1}$, $\mathbf{C} = R(\mathbf{x}_i - \mathbf{x}_j)$ as an $n \times n$ matrix of correlations amongst the elements of ξ, and $\mathbf{c}_{n+1} = R(\mathbf{x}_i - \mathbf{x}_{n+1})$ as an $n \times 1$ vector of correlations between $\mathbf{x}_{n+1}$ and the elements of ξ.

The parameters $\boldsymbol{\beta}$ and σ_z^2, which are defined above, and the $q \times 1$ vector of correlation parameters $\boldsymbol{\theta} = (\theta_1, ..., \theta_q)^T$ are unknown. We first obtain a joint conditional distribution for Y_{n+1} and $\mathbf{Y}_n$ given these unknown parameters:

$$f\begin{pmatrix} Y_{n+1} \\ \mathbf{Y}_n \end{pmatrix} \Bigg| (\boldsymbol{\beta}, \sigma_z^2, \boldsymbol{\theta}) \sim N_{n+1}\left[\begin{pmatrix} \mathbf{f}_{n+1}^T \\ \mathbf{F} \end{pmatrix} \boldsymbol{\beta}, \sigma_z^2 \begin{pmatrix} 1 & \mathbf{c}_{n+1}^T \\ \mathbf{c}_{n+1} & \mathbf{C} \end{pmatrix}\right].$$

If all three of $\boldsymbol{\beta}$, σ_z^2 and $\boldsymbol{\theta}$ are unknown, the posterior predictive distribution cannot be determined analytically. If, however, the correlation parameters $\boldsymbol{\theta}$ are treated as being known, the predictive distribution can be derived. This is done by finding the conditional density,

$$f(Y_{n+1} | \sigma_z^2, \boldsymbol{\beta}, \mathbf{Y}_n)\,,$$

and integrating out the two unknown variables. To obtain this we need a conjugate prior distributions for $\boldsymbol{\beta}|\sigma_z^2$: a multivariate normal with known mean $\mathbf{b}_0$ and variance-covariance matrix $\sigma_z^2 \mathbf{V}_0$. A conjugate prior is also required for σ_z^2: this is a scaled-inverse-chi-squared distribution with parameters $(\nu_0, c_0/\nu_0)$, where ν_0 is the degrees of freedom and c_0 is a constant. This can equivalently be viewed as an inverse-gamma $(\nu_0/2, c_0/2)$ distribution, with density given by

$$f(\sigma_z^2) = \frac{(c_0/2)^{(\nu_0/2)}}{\Gamma(\nu_0/2)} (\sigma_z^2)^{-(\nu_0/2)-1} \exp\left\{ -\frac{c_0}{2\sigma_z^2} \right\}.$$

The details of the derivation of the posterior predictive distribution are given in Chapter 5 of Santner et al. (2003), with the result that the posterior predictive distribution is a non-standardised t-distribution. In one dimension, this is a function of three parameters: the degrees of freedom ν, location parameter μ, and scale parameter σ^2. It is denoted as $T_1(\nu, \mu, \sigma^2)$ and has the density function

$$f(w) = \frac{\Gamma[(\nu+1)/2]}{\Gamma(\nu/2)\pi^{(1/2)}\sqrt{\nu\sigma^2}} \left(1 + \frac{(w-\mu)^T(\sigma^2)^{-1}(w-\mu)}{\nu} \right)^{(\nu+1)/2}.$$

The posterior predictive distribution for Y_{n+1} has the form

$$f(Y_{n+1} | \mathbf{Y}_n) \sim T_1(\nu^*, \mu^*, \sigma^{*2})\,, \tag{2.4}$$

where $\nu^* = \nu_0 + n$,

$$\mu^* = \mathbf{f}_{n+1}^T \hat{\boldsymbol{\beta}} + \mathbf{c}_{n+1}^T \mathbf{C}^{-1} (\mathbf{y}_n - \mathbf{F}\hat{\boldsymbol{\beta}})\,, \tag{2.5}$$

and

$$\sigma^{*2} = \frac{Q_1}{\nu^*}\left\{1 - (\mathbf{f}_{n+1}^T, \mathbf{c}_{n+1}^T)\begin{bmatrix} -\mathbf{V}_0^{-1} & \mathbf{F}^T \\ \mathbf{F} & \mathbf{C} \end{bmatrix}\begin{pmatrix} \mathbf{f}_{n+1} \\ \mathbf{c}_{n+1} \end{pmatrix}\right\}.$$

In the above equations, we have

$$\hat{\boldsymbol{\beta}} = (\mathbf{V}_0^{-1} + \mathbf{F}^T\mathbf{C}^{-1}\mathbf{F})^{-1}(\mathbf{V}_0^{-1}\mathbf{b}_0 + \mathbf{F}^T\mathbf{C}^{-1}\mathbf{y}_n)\,, \tag{2.6}$$

a Bayesian form of the generalised least squares regression estimator for the unknown regression coefficients $\boldsymbol{\beta}$. We also have

$$Q_1 = c_0 + \mathbf{y}_n^T\left[\mathbf{C}^{-1} - \mathbf{C}^{-1}\mathbf{F}(\mathbf{F}^T\mathbf{C}^{-1}\mathbf{F})^{-1}\mathbf{F}^T\mathbf{C}^{-1}\right]\mathbf{y}_n + H_1\,,$$

where

$$H_1 = (\mathbf{b}_0 - \hat{\boldsymbol{\beta}})^T(\mathbf{V_0} + [\mathbf{F}^T\mathbf{C}^{-1}\mathbf{F}]^{-1})^{-1}(\mathbf{b}_0 - \hat{\boldsymbol{\beta}})\,.$$

The conjugate prior distributions require several parameters to be estimated, and choosing any specific value for them can be thought of as an informative decision. The information required to make sensible informative choices for the prior distributions is rarely available in advance. A more practical alternative is to use non-informative or weak priors instead. For the regression coefficients $\boldsymbol{\beta}$, a non-informative conjugate prior is the constant

$$f(\boldsymbol{\beta}|\sigma_z^2) = 1\,.$$

The weak prior for the process variance σ_z^2 is the Jeffreys prior, introduced by Jeffreys (1961):

$$f(\sigma_z^2) = \frac{1}{\sigma_z^2}\,.$$

Per Rasmussen and Williams (2006), Chapter 2, these distributions can be viewed as limiting cases of their strong counterparts as the prior variance of the parameters is infinitely large. Neither of these prior distributions is a proper distribution, but this is not a concern as the resulting posterior distribution is proper; further details can be found in Santner et al. (2003), Chapter 5. Using these priors, the posterior predictive distribution is again gives a non-standardised t-distribution, but now with the parameters $\nu^* = n - p$,

$$\mu^* = \mathbf{f}_{n+1}^T\hat{\boldsymbol{\beta}} + \mathbf{c}_{n+1}^T\mathbf{C}^{-1}(\mathbf{y}_n - \mathbf{F}\hat{\boldsymbol{\beta}})\,, \tag{2.7}$$

and

$$\sigma^{*2} = \frac{Q_2}{\nu^*}\left\{1 - (\mathbf{f}_{n+1}^T, \mathbf{c}_{n+1}^T)\begin{bmatrix} -\mathbf{0} & \mathbf{F}^T \\ \mathbf{F} & \mathbf{C} \end{bmatrix}\begin{pmatrix} \mathbf{f}_{n+1} \\ \mathbf{c}_{n+1} \end{pmatrix}\right\},$$

where

$$Q_2 = \mathbf{y}_n^T \Big[\mathbf{C}^{-1} - \mathbf{C}^{-1}\mathbf{F}(\mathbf{F}^T\mathbf{C}^{-1}\mathbf{F})^{-1}\mathbf{F}^T\mathbf{C}^{-1} \Big] \mathbf{y}_n \,,$$

and

$$\hat{\boldsymbol{\beta}} = (\mathbf{F}^T\mathbf{C}^{-1}\mathbf{F})^{-1}(\mathbf{F}^T\mathbf{C}^{-1}\mathbf{y}_n) \,. \tag{2.8}$$

Also of interest is the simpler case in which the process variance σ_z^2 is treated as known. In this case, a conjugate prior distribution is required for $\boldsymbol{\beta}$ only; this is, as in the case of unknown process variance, a multivariate normal distribution with mean $\mathbf{b}_0$, but its variance-covariance matrix is now $\tau^2\mathbf{V}_0$ for a constant τ^2. Following a method presented by several authors including Santner et al. (2003), Chapter 5, the resulting posterior predictive distribution is a one-dimensional normal distribution,

$$f(Y_{n+1}|\mathbf{Y}_n) \sim N(\mu^*, \sigma^{*2}) \,, \tag{2.9}$$

with parameters

$$\mu^* = \mathbf{f}_{n+1}^T\hat{\boldsymbol{\beta}} + \mathbf{c}_{n+1}^T\mathbf{C}^{-1}(\mathbf{y}_n - \mathbf{F}\hat{\boldsymbol{\beta}}),$$

and

$$\sigma^{*2} = \sigma_z^2 \left\{ 1 - (\mathbf{f}_{n+1}^T, \mathbf{c}_{n+1}^T) \begin{bmatrix} -\frac{\sigma_z^2}{\tau^2}\mathbf{V}_0^{-1} & \mathbf{F}^T \\ \mathbf{F} & \mathbf{C} \end{bmatrix} \begin{pmatrix} \mathbf{f}_{n+1} \\ \mathbf{c}_{n+1} \end{pmatrix} \right\}.$$

The posterior estimate of the regression coefficients, $\hat{\boldsymbol{\beta}}$, is now given by

$$\hat{\boldsymbol{\beta}} = \left(\frac{\mathbf{V}_0^{-1}}{\tau^2} + \frac{\mathbf{F}^T\mathbf{C}^{-1}\mathbf{F}}{\sigma_z^2} \right)^{-1} \left(\frac{\mathbf{V}_0^{-1}\mathbf{b}_0}{\tau^2} + \frac{\mathbf{F}^T\mathbf{C}^{-1}\mathbf{y}_n}{\sigma_z^2} \right),$$

Again, the choice of the parameters of the prior distribution imposes strong information on the posterior predictive distribution, so a non-informative approach is also possible by setting the distribution for the regression coefficients to $f(\boldsymbol{\beta}) = 1$. This again yields a normal distribution for the emulator output, with parameters

$$\mu^* = \mathbf{f}_{n+1}^T\boldsymbol{\mu}_\beta + \mathbf{c}_{n+1}^T\mathbf{C}^{-1}(\mathbf{y}_n - \mathbf{F}\hat{\boldsymbol{\beta}})$$

and

$$\sigma^{*2} = \sigma_z^2 \left\{ 1 - (\mathbf{f}_{n+1}^T, \mathbf{c}_{n+1}^T) \begin{bmatrix} -\mathbf{0} & \mathbf{F}^T \\ \mathbf{F} & \mathbf{C} \end{bmatrix} \begin{pmatrix} \mathbf{f}_{n+1} \\ \mathbf{c}_{n+1} \end{pmatrix} \right\}, \tag{2.10}$$

where $\hat{\boldsymbol{\beta}}$ is defined in (2.8).

In the simplest case, the regression coefficients $\boldsymbol{\beta}$ can also be considered to be known, meaning that no prior distributions are required. Then, the posterior predictive distribution is a normal distribution with mean

$$\mu^* = \mathbf{f}_{n+1}^T\boldsymbol{\beta} + \mathbf{c}_{n+1}^T\mathbf{C}^{-1}(\mathbf{y}_n - \mathbf{F}\boldsymbol{\beta})\,,$$

and variance

$$\sigma^{*2} = \sigma_z^2(1 - \mathbf{c}_{n+1}^T\mathbf{C}\mathbf{c}_{n+1})\,. \tag{2.11}$$

It is useful to demonstrate prediction from a GP emulator using a simple one-dimensional example. Consider a simulator defined by the function

$$y = x^4e^{-x} + \sin(2x)\,, -3.5 \leq x \leq 6.5\,. \tag{2.12}$$

We assume that the equation which defines the simulator is unknown, and construct a GP emulator based on four simulator runs for different values of x, with the design ξ being defined by the vector $\xi = (3.5, 4.5, 5.5, 6.5)^T$. A constant regression term and a Matérn correlation function with smoothness parameter $\omega = 5/2$ are used in the emulator. The GP emulator is then used to predict the simulator output at 200 equally-spaced prediction points across the range of x.

The resulting predictions are plotted in Figure 2.1. The true simulator output is shown in the plot as a solid black line, the design points in red, the mean prediction from the emulator in blue, and the bounds of a 95% prediction interval from the emulator in green. The plot demonstrates that the emulator mean is identical to the true simulator output at the design points, but may differ from it at untested inputs. The uncertainty in the emulator estimates is zero at the design points where the simulator output is known, and increases with the distance from the nearest design point.

Computational issues can arise when $\mathbf{C}$, the matrix of correlations between the design points, is inverted. If the correlation between any pair of design points is very large, or if the correlation between all pairs of design points is very small, $\mathbf{C}$ can be close to singular. To overcome this, it is common to add a small error term δ, called a nugget, to the diagonal of the matrix:

$$\mathbf{C} = \mathbf{C} + \delta I\,.$$

This definition of the nugget follows that of Andrianakis and Challenor (2012). Other authors, including Gramacy and Lee (2012), define the nugget as a random term with variance δ/σ_z^2. We can interpret the use of a nugget as introducing a small amount of uncertainty into the simulator output at the design points. This ensures that $\mathbf{C}$ can be inverted without difficulty. In Figure 2.1, the green lines would no longer meet at the design points, but would lie a very small distance away from the true value. A

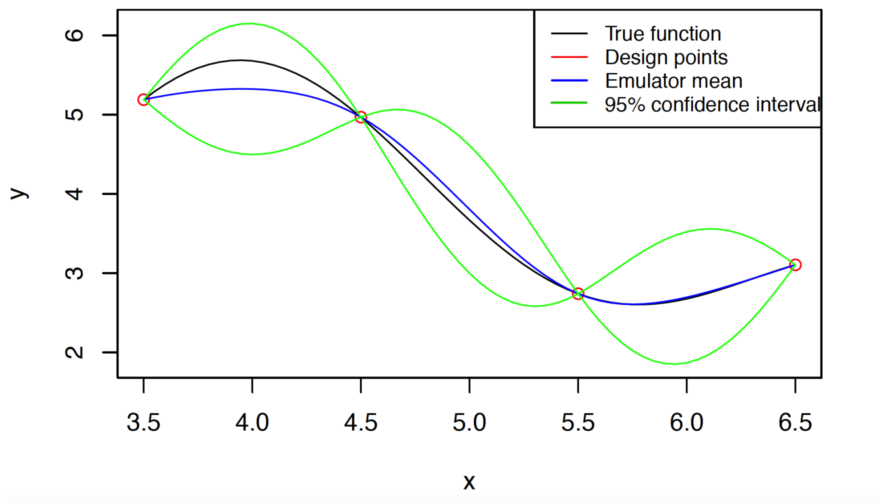

Figure 2.1: Prediction from a Gaussian process emulator for the simulator defined in equation (2.12).

fuller treatment of the use of a nugget was conducted by Andrianakis and Challenor (2012), considering its effect on prediction from the resulting emulator under various conditions. In addition, Gramacy and Lee (2012) show that using a non-zero nugget in a GP emulator can have a positive effect on statistical properties including coverage and predictive accuracy. For computational reasons, we makes use of a nugget in most of the emulators constructed in our work.

GP emulation can occasionally produce poor predictions for the true simulator output. The use of a stationary Gaussian process prior assumes that the correlation in each dimension depends only on distance, not on location; that the simulator is smooth and continuous; and that the residuals from our emulator estimate are equally likely to lie on either side of the estimate in each dimension. These assumptions may not be true for some simulators. The emulator also requires several parameters to be chosen, either directly or through the specification of prior distributions for Bayesian inference, which can introduce a conflict with the data. These issues are discussed in detail by O'Hagan (2006), and diagnostics which can be used to test the performance of an emulator are introduced in the same paper; more recently, Bastos and O'Hagan (2009) also considered diagnostics for a GP emulator.

Several software implementations for Gaussian process emulation exist in the `R` language. The most popular package is '`DiceKriging`' (Roustant et al., 2012), which includes many methods for fast calculation of emulators using a partially Bayesian framework, with maximum likelihood estimation for the correlation parameters. '`DiceKriging`' supports two families of correlation functions: Matérn with $\omega = 5/2$ or $\omega = 3/2$, and power-exponential with $0 < \alpha \leq 2$, including the special case of the Gaussian correlation function. Another package, '`mlegp`' (Dancik and Dorman, 2008), implements the same methods for the Gaussian correlation function only.

2.3 Unknown parameters of the Gaussian process emulator

It is not usually possible to know in advance which parameter values will lead to an appropriate GP emulator. As demonstrated above, it is possible to integrate out some of these parameters for certain choices of prior distribution, but this cannot be done for the correlation parameters, and it is sometimes desirable to estimate the process variance and regression coefficients directly. Additionally, the appropriate size of the nugget of the Gaussian process cannot typically be determined until the emulator is fitted. There are two main ways to estimate the unknown parameters.

2.3.1 Plug-in approach

The simpler method is a plug-in approach, in which a single value of each unknown parameter is found and used throughout the following analysis in place of the full distribution of the parameter(s). This has the advantage of being able to treat the parameter as a single known value, allowing the theoretical results derived above to be used directly. It is also less computationally intensive than other approaches.

A common method to estimate the unknown parameters by a single value is maximum likelihood estimation (MLE) , in which the value of the parameters is chosen to maximise their joint likelihood given the observed data. This is an intuitively sensible choice, since it ensures that the parameters are chosen in a way which makes the observed data most likely to have been reached. Its use in GP emulation for computer experiments dates to the beginnings of the field itself, including in the work of Sacks et al. (1989), and it remains the most widespread choice.

In practice, maximising the joint likelihood of the parameters is difficult, so they are typically estimated separately (see Chapter 5 of Fang et al., 2006). Maximum likelihood estimation of the of the regression coefficients, $\boldsymbol{\beta}$, and process variance, σ_z^2, is particularly straightforward as the MLE for these parameters is available in closed form. For the regression coefficients, the MLE is the generalised least-squares regression estimator introduced in equation (2.8). The MLE of the process variance is

$$\hat{\sigma}_z^2 = \frac{1}{n}(\mathbf{y}_n - \mathbf{F}\hat{\boldsymbol{\beta}})^T \mathbf{C}^{-1}(\mathbf{y}_n - \mathbf{F}\hat{\boldsymbol{\beta}})\,.$$

The MLE of the correlation parameters $\boldsymbol{\theta}$ is not available in closed form, so this must be found using an iterative numerical method. The matrix $\mathbf{C}$, which appears in the MLEs for $\boldsymbol{\beta}$ and σ_z^2, depends on $\boldsymbol{\theta}$; it is thus necessary to update the estimates of $\boldsymbol{\beta}$ and σ_z^2 after updating $\boldsymbol{\theta}$ using the chosen numerical method. An algorithm to do this is presented in Chapter 5 of Fang et al. (2006) .

Alternative approaches to simple maximum likelihood estimation also exist. The restricted maximum likelihood (REML) method, introduced by Patterson and Thompson (1971) for incomplete block design experiments, computes less biased estimates for variance and covariance/correlation parameters than MLE. This is achieved by maximising the likelihood of a set of linearly independent combinations of the observed data instead of the data itself. As for MLE, the REML estimate of σ_z^2 is available in closed form, while the estimate of $\boldsymbol{\theta}$ is not. Chapter 3 of Santner et al. (2003) provides further details.

The likelihood function for the correlation parameters $\boldsymbol{\theta}$ can often be very flat in practice, meaning there is a large variance in the maximum likelihood or REML estimates. This can occur even in very simple examples, for example the one-dimensional sinusoidal function given by Li and Sudjianto (2005). The same authors proposed a solution using a penalised likelihood, in which a penalty function is added to the likelihood before maximisation takes place. This can have the effect of increasing the variation around the maximum, reducing the variance of the resulting estimate. The suggested penalty function is the smoothly clipped absolute deviation (SCAD) penalty (Fan, 1997).

From a Bayesian perspective, MLE can be viewed as a special case of the more general maximum a posteriori estimation (MAP). Instead of just maximising the likelihood, this method maximises a posterior distribution derived from both the likelihood and a prior distribution. This is also referred to as posterior mode estimation, and has been considered as a method to obtain the parameters of a GP emulator by several authors including Santner et al., 2003, Chapter 3, and Gu et al., 2018. The choice of the prior distribution for the parameters is an important one, since this will significantly effect the posterior distribution to be maximised. Maximum likelihood estimation corresponds to the use of a uniform prior distribution on the parameters of interest. This is in some ways a natural choice for parameters of a GP emulator, about which little may be known in advance. When non-uniform conjugate priors are chosen to allow the regression coefficients and process variance to be integrated out, however, these should be taken account of in the estimation of the correlation parameters. In addition, Gu et al. (2018) presented an argument for using non-uniform priors on the correlation parameters based on the robustness of the parameter estimates, as this is less likely to lead to $\mathbf{C}$ being close to singular or to the identity matrix $\mathbf{I}$.

It is worth reviewing parameter estimation in existing `R` packages for GP emulation, since our code makes use of these packages in places. `'mlegp'` uses exclusively maximum likelihood estimation for all of the unknown parameters of the emulator. `'DiceKriging'` also supports this approach, but provides additional options. Penalised maximum likelihood estimation with a SCAD penalty function is also offered. The parameters may also be entered by the user directly, which allows other methods to be used.

One oddity concerning the two packages is the scaling of the correlation parameters of the power-exponential correlation function. While `‘mlegp’` requires $\alpha = 2$ and uses the form given in equation (2.2), `‘DiceKriging’` allows α to vary and uses the alternative parameterisation

$$R(\mathbf{h}) = \exp\left[-\sum_{j=1}^{q} \frac{1}{2}\left(\frac{h_j}{b_j}\right)^{\alpha}\right].$$

In our work, the form given in equation (2.1) is used. However, since we use the `‘DiceKriging’` package to estimate correlation parameters in some of our examples, the values returned by the package must first be transformed to our preferred scale. Parameter estimation in the following chapters is conducted using MLE where applicable, and MAP with conjugate priors for the regression coefficients and process variance where these parameters are integrated out. It should be noted that if MAP with a non-uniform reference prior on the correlation parameters is considered, Gu et al. (2018) suggest avoiding the parameterisation given in equation (2.1), but this is not an issue for us as reference priors are not used in this thesis.

None of the plug-in estimation methods described above allow the form of the maximiser for the correlation parameters to be determined analytically, so this requires numerical optimisation. This can be a difficult problem, as the function to be optimised may be either extremely flat or have many local maxima. There are several algorithms which could be used, many of which are listed in Chapter 3 of Santner et al. (2003). The approach taken by `‘DiceKriging’` is based on a combination of the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm, an iterative method based on local search and gradient descent, and a genetic algorithm for global search; it is described in further detail by Roustant et al. (2012). Even with this combined method, it is still common for only a local optimum to be found, but this can be partially overcome by running the optimisation algorithm from many different starting locations within the space of the correlation parameters.

The main weakness of a plug-in approach to the unknown parameters is that it ignores a source of uncertainty in predictions from the emulator. By treating the unknown parameters as single values instead of a distribution of possible values, the uncertainty in them is not captured in the analysis that follows, leading to potentially low variance bounds and overconfidence in the resulting predictions. In the case of the nugget, which exists only for computational purposes, this is not a problem provided a reasonable value is chosen. For the correlation parameters, and (where relevant) the process variance and regression coefficients, it is more of a concern. Despite this, due to its computational simplicity and widespread prior use in both literature and software implementation, a plug-in method based on maximum likelihood estimation forms the basis of our work in the remainder of the thesis. The following subsection discusses an alternative approach which accounts for the uncertainty in the estimates of the parameters in its predictions, but which was not taken forward due to the increased computational resources required.

2.3.2 Markov chain Monte Carlo

A more complex alternative is a Bayesian approach in which a sample of values for the parameters is generated using Markov chain Monte Carlo (MCMC). This is typically done only for the correlation parameters $\boldsymbol{\theta}$, as the regression coefficients and process variance can instead be integrated out as described in Section 2.2. MCMC is a set of methods which are designed to sample from a probability distribution by constructing and sampling from a Markov chain with the target probability distribution as its limiting distribution. Using this approach, it is possible to sample from the distribution without knowing its constant of proportionality; we require only a density proportional to that of the target. Detailed coverage of the topic can be found in books by several authors, including Gamerman and Lopes (2006).

For a Markov chain with a continuous state space S, let A be a subset of S. Let $\boldsymbol{\theta}$ be the vector of parameters of interest, and $\boldsymbol{\theta}^{(j)}$ be the value of the parameters of interest after j timesteps. A Markov chain is called time-homogeneous if its transition probabilities do not depend on the number of steps made so far; the equality

$$P[\boldsymbol{\theta}^{(M+1)} \in A|\boldsymbol{\theta}^{(M)}, \boldsymbol{\theta}^{(M-1)}, ..., \boldsymbol{\theta}^{(0)}] = P[\boldsymbol{\theta}^{(1)} \in A|\boldsymbol{\theta}^{(0)}]$$

holds for all $M \in 0, 1, 2, \ldots$, and $A \in S$. It is then possible to define a transition function $P(\phi, A)$, called the kernel of the chain, which states the probability of moving from a state ϕ to a subset $A \in S$ in a single step. We require that the kernel is a probability distribution over S for any ϕ, and that it is possible to evaluate it for any choices of ϕ and A. The kernel can be used to generate the next state of the chain. For two states ψ, ϕ we may choose to split the kernel $P(\psi, \phi)$ into a transition kernel $q(\psi)$, which determines a proposal ϕ for the next state from the kernel, and an acceptance probability $a(\psi, \phi)$, which determines how often the proposal is accepted. If the proposal is rejected the chain remains in the current state for a further timestep.

A Markov chain is irreducible if it is possible to reach any subset of the state space from any other in a finite number of steps. The expected number of steps in which the chain will return to the subset is called the mean recurrence time; a subset is positive recurrent if its mean recurrence time is finite.

The stationary distribution, π, of a Markov chain is a distribution over its state space such that if the probability of being in each state of the chain follows the distribution π at any timestep of the chain, it will still follow π at the next timestep. This can be expressed mathematically as

$$\int \pi(\psi)P(\psi, \phi)\mathrm{d}\psi = \pi(\psi)\,.$$

Consider an irreducible, time-homogeneous Markov chain in which every subset of the state space is positive-recurrent. A result given in several sources including Nummelin (1984) states that if this chain has a stationary distribution π, the distribution of the states converges to π as the number of steps increases, irrespective of the initial state. π is called the limiting distribution (sometimes equilibrium distribution or invariant distribution) of the Markov chain.

A time-homogeneous Markov chain with limiting distribution π is reversible if

$$\pi(\psi)P(\psi,\phi) = \pi(\phi)P(\phi,\psi) \quad \forall\, \phi,\psi \in S \tag{2.13}$$

which is called the detailed balance equation, and implies that the probability of making a transition from ψ to ϕ is the same as that of transitioning from ϕ to ψ under the limiting distribution. The converse is also true: if a distribution D satisfies equation (2.13) for a Markov chain with kernel $P(\psi,\phi)$, then D is the limiting distribution of the chain.

MCMC methods exploit this to create a Markov chain with the probability distribution we wish to sample from as its limiting distribution. To be certain that the properties of the chain approximate those of the target distribution, we require the chain to be aperiodic. The period of a subset A of S is the greatest common divisor of all possible numbers of steps in which the chain can return to A. A subset A is aperiodic if it has period 1; a Markov chain is aperiodic if this is true for all $A \in S$. Under the conditions of irreducibility, aperiodicity and positive recurrence of states, a sample approximation to a function $t(\boldsymbol{\theta})$,

$$t'_M = \frac{1}{M}\sum_{j=1}^{n} t[\boldsymbol{\theta}^{(j)}]\,,$$

converges with probability 1 to the expectation of $t(\boldsymbol{\theta})$ as $M \to \infty$. This means that for a sufficiently large sample size M, per sources such as Gamerman and Lopes (2006), the sample average t'_M can be used as an estimate for $t(\boldsymbol{\theta})$.

To use MCMC to sample from the distribution $D(\boldsymbol{\theta})$ of the parameters of interest, we must construct a chain with $D(\boldsymbol{\theta})$ as its limiting distribution. This is achieved by choosing a kernel $P(\psi,\phi)$ which satisfies (2.13) for $D(\boldsymbol{\theta})$. The longest-established and most common choice is the Metropolis-Hastings algorithm , which in its earliest form was proposed by Metropolis et al. (1953) and was later generalised by Hastings (1970). The transition kernel $\phi = q(\psi)$ is a random walk transition,

$$\phi = \psi + \mathbf{w}\,, \tag{2.14}$$

where $\mathbf{w}$ is a randomly generated vector with probability density f_w. A Markov chain with a random walk transition kernel is reversible if f_w is symmetric about 0, so several options for this density are possible; the most common is a multivariate normal distribution. The acceptance probability $a(\psi, \phi)$ is

$$\begin{aligned} a(\psi, \phi) &= min\Big\{1, \frac{D(\phi)q(\phi)}{D(\psi)q(\psi)}\Big\} \\ &= min\Big\{1, \frac{D(\phi)}{D(\psi)}\Big\}, \end{aligned}$$

where the second equality holds since $q(\psi)$ is reversible. This is equivalent to

$$a(\psi, \phi) = min\Big\{1, \frac{f_D(\phi)}{f_D(\psi)}\Big\}, \tag{2.15}$$

where $f_D(\boldsymbol{\theta})$ is a function proportional to the target density $D(\boldsymbol{\theta})$, as the normalising constant in the density appears in both the numerator and denominator of the second term of the acceptance probability and thus cancels out. Algorithm 1 sets out the steps of the Metropolis-Hastings algorithm formally.

Algorithm 1: Metropolis-Hastings algorithm

```
Input: Initial value $\boldsymbol{\theta}^{(0)}$; target sample size $M$
Output: Sample matrix $\boldsymbol{\Theta} = (\boldsymbol{\theta}^{(1)}, ..., \boldsymbol{\theta}^{(M)})^T$ from target density $D(\boldsymbol{\theta})$
begin
    for $j \leftarrow 1$ to $M$ do
        $\phi \leftarrow$ sample $(q(\boldsymbol{\theta}^{(j-1)}))$ ;          // $q()$ defined in (2.14)
        $p \leftarrow a(\boldsymbol{\theta}^{(j-1)}, \phi)$ ;                  // $a()$ defined in (2.15)
        $u \leftarrow$ sample (unif$(0, 1)$) ;
        if $u < p$ then $\boldsymbol{\theta}^{(j)} = \phi$;
        else $\boldsymbol{\theta}^{(j)} = \boldsymbol{\theta}^{(j-1)}$;
    end
end
```

When a multivariate normal distribution is used to generate the proposals, the choice of the covariance matrix can have a large effect on the algorithm's performance. The covariance matrix can be either a scalar multiple of the identity matrix or of an estimated matrix of correlations between the dimensions of the target density. In either case the scalar which multiplies this matrix, called the variance parameter or v, must be carefully chosen. If it is too large, the transitions proposed are also large and are thus unlikely to be accepted as they will frequently take the chain to regions of the state space where the target density is very low. Equally, a very small choice for v means that the proposed transitions are also small, so the chain will take a long time to explore the whole of the space. In extreme cases, the chain can remain in one region of the state space where the density is high while failing to reach another similar region, as few or no proposals will be large enough to make this jump. These issues were discussed in detail by Roberts et al. (1997), with the conclusion that a good value for v can be determined from the probability of a transition being accepted. The paper demonstrates that, as the dimensionality of the state space tends to infinity, the optimal acceptance probability under certain conditions tends to 0.234. In one dimension, the optimal acceptance probability was found to be close to 0.5.

Instead of a random walk, other transition kernels are possible. For example, Hamiltonian Monte Carlo (HMC) has a structure similar to that of the Metropolis-Hastings algorithm, but the Markov chain used is defined differently (Neal, 2011). Proposed transitions are generated using random sampling from a multivariate distribution defined by a process called Hamiltonian dynamics, which originated in physics as a 19th-century reformulation of classical mechanics. This technique, introduced by Duane et al. (1987) in the field of molecular simulation, allows much larger transitions to be made than random walk proposals.

For MCMC sampling to take place for the correlation parameters, we require a density proportional to that of $f(\boldsymbol{\theta}|\mathbf{Y}_n)$. Using the same informative conjugate prior distributions for $\boldsymbol{\beta}|\sigma_z^2$ and σ_z^2 as in Section 2.2, and following a method based on deriving marginal distributions using integration presented in the context of Bayes linear models by Banerjee (2010), the proportional density can be derived to be

$$f(\boldsymbol{\theta}|\mathbf{Y}_n) \propto \frac{[c_0 + \mathbf{e}^T(\mathbf{C}^{-1} - \mathbf{C}^{-1}\mathbf{F}(\mathbf{V}_0^{-1} + \mathbf{F}^T\mathbf{C}^{-1}\mathbf{F})^{-1}\mathbf{F}^T\mathbf{C}^{-1})\mathbf{e}]^{-\frac{\nu_0+n}{2}}}{|\mathbf{V}_0^{-1} + \mathbf{F}^T\mathbf{C}^{-1}\mathbf{F}|^{1/2}|\mathbf{C}|^{1/2}},$$

where

$$\mathbf{e} = \mathbf{y}_n - \mathbf{F}\mathbf{b}_0 .$$

If non-informative priors are instead chosen for $\boldsymbol{\beta}|\sigma_z^2$ and σ_z^2 (again as in Section 2.2), the result is

$$f(\boldsymbol{\theta}|\mathbf{Y}_n) \propto \frac{[(\mathbf{y}_n)^T(\mathbf{C}^{-1} - \mathbf{C}^{-1}\mathbf{F}(\mathbf{F}^T\mathbf{C}^{-1}\mathbf{F})^{-1}\mathbf{F}^T\mathbf{C}^{-1})(\mathbf{y}_n)]^{-n/2}}{|\mathbf{F}^T\mathbf{C}^{-1}\mathbf{F}|^{1/2}|\mathbf{C}|^{1/2}}.$$

Computing the actual value of these densities can be a challenge: as the number of design points n increases, the power $-(\nu_0 + n)/2$ or $-n/2$ becomes large and negative, with the result that any value raised to this power is extremely close to zero, causing computational difficulties. This obstacle can be removed by working with the log-likelihood instead of the density function of the pure likelihood. With informative priors on $\boldsymbol{\beta}|\sigma_z^2$ and σ_z^2, this is given by

$$\begin{aligned} l(\mathbf{y}_n) \propto K_1 &- \frac{1}{2}\log\left|(\mathbf{C} + \mathbf{F}\mathbf{V}_0\mathbf{F}^T)\right| \\ &- \left(\frac{\nu_0 + n}{2}\right)\log\left[1 + \frac{1}{c_0}\mathbf{e}^T(\mathbf{C} + \mathbf{F}\mathbf{V}_0\mathbf{F}^T)^{-1}\mathbf{e}\right], \end{aligned}$$

where K_1 is a constant with respect to $\boldsymbol{\theta}$. Similarly, with non-informative priors on $\boldsymbol{\beta}|\sigma_z^2$ and σ_z^2, the log-likelihood is

$$\begin{aligned} l(\mathbf{y_n}) \propto K_2 - \frac{1}{2}\log|\mathbf{F}^T\mathbf{C}^{-1}\mathbf{F}| - \frac{1}{2}\log|\mathbf{C}| \\ - \frac{n}{2}\log\left[(\mathbf{y_n})^T(\mathbf{C}^{-1} - \mathbf{C}^{-1}\mathbf{F}(\mathbf{F}^T\mathbf{C}^{-1}\mathbf{F})^{-1}\mathbf{F}^T\mathbf{C}^{-1})(\mathbf{y_n})\right], \end{aligned}$$

with K_2 constant with respect to $\boldsymbol{\theta}$.

In practise, the determinants in these equations can be small enough to be computationally zero, leading to problems when their logarithm is taken. This can be overcome by recalling that the determinant of a matrix is the product of its eigenvalues, which we denote by $\lambda_1, ..., \lambda_n$. Since the logarithm of a product of terms is equal to the summation of the logarithms of these terms, the logarithm of the determinant of a matrix Λ can thus be rewritten as

$$\log|\Lambda| = \sum_{i=1}^{n} \log \lambda^{(i)} ,$$

which is more computationally stable as it does not involve the product of the eigenvalues.

Using MCMC for the correlation parameters of the GP emulator allows a fully Bayesian analysis which captures all sources of uncertainty in an emulator, leading to more robust predictions. It is however more computationally intensive, and reduces the use that can be made of theoretical results, instead requiring large simulations from the emulator using the MCMC sample for the correlation parameters. In the early stages of our work, MCMC was considered as the better approach and was implemented for emulators for a single computational model. For chains of models, this was not taken forward for two reasons: speed of execution of the linked emulator, and the wish to make use of theoretical results which will be presented in Chapter 3. Incorporating MCMC into the construction and analysis of linked emulators nonetheless remains a potentially valuable avenue for future research.

2.4 Extensions of the GP emulator

Although Gaussian process emulation is a powerful modelling tool, there are cases it cannot cover without further extension. The assumption of stationarity in the GP emulator is a strong one, which will often not be met in the real world; it is common to encounter models with much larger variation (and thus lower correlation) between nearby points in some parts of the input space than others. One solution to this is the use of a treed Gaussian process, in which treed partitioning is used to determine regions of approximate stationarity, allowing the fitted emulator to use different stationary GPs in different regions of the design space. Gramacy and Lee (2008) provides further details on this approach. The R package `'tgp'` (Gramacy and Taddy, 2010) provides a software implementation of this approach. Unlike the packages for standard GP emulation discussed above, it allows the use of Markov chain Monte Carlo to obtain a sample for the correlation parameters.

Categorical input variables are another complicating factor for GP emulation due to their effect on the correlation function of the Gaussian process. If a model with one or more continuous inputs also takes an input with only two levels, any two input configurations with different levels of the binary variable are a fixed distance apart in the space of this input, at the maximum possible distance. A standard multiple-input correlation function such as the product power-exponential or product Matérn will give a scale parameter in this dimension which returns correlations close to zero if the input has a substantial effect on the output, rendering correlations in the other dimensions irrelevant and the overall correlation obtained not particularly meaningful. Qian et al. (2009) attempt to address this by developing bespoke correlation functions to handle a mixture of categorical and continuous input variables.

In many real processes to which fitting an emulator would be useful - and, increasingly, in many computational models - repeating a measurement at the same input configurations will not return the same output. Such a model is called stochastic ; if the variance depends on the inputs, the model is additionally heteroscedastic. A standard GP emulator is not appropriate for such a model, since it will assume that the variance at a design point is only that of the nugget (which may be unrealistically low), and that the variance is the same at every design point (when it is likely to vary across the input space). Instead, the stochasticity and heteroscedasticity in the output should be accounted for directly.

Goldberg et al. (1998) propose a way to do this: fit a stationary GP emulator to the mean of the simulator response, and a second independent stationary GP emulator to the variance of the response, with MCMC sampling for both the parameters of the Gaussian processes and the posterior distribution of the variance. Kersting et al. (2007) follow a similar approach, but with maximum likelihood estimation instead of MCMC, and demonstrate that the method is effective for several data sets. More recently, the R package `'hetGP'` has provided a software implementation of this; the associated paper of Binois et al. (2018) considers in detail the problem of experimental design for a heteroscedastic GP emulator, which is more complicated than for the traditional case as replication is required to determine the nature of the variance of the response.

Finally, the framework described above holds only if the computer model of interest returns a single output. Simulators with multiple outputs can be handled in a variety of ways. If the number of outputs is small, a separate emulator may be built for each output; this is done by Kyzyurova et al. (2018) among others. For high-dimensional outputs, this becomes very computationally intensive. A specific example of relevance to our application is dispersion modelling. The output in this case is typically the concentration of a contaminant across a two-dimensional grid, often over time. Several approaches have been suggested to deal with this by various authors.

In addition to several separate emulators, Conti and O'Hagan (2010) consider two methods to deal with a simulator which returns a vector of outputs over time. The first is to explicitly build an emulator for all of the outputs simultaneously. The output of the computer experiment is now multi-dimensional, and the joint posterior predictive distribution for the simulator output at an arbitrary point in the input space given the parameters of the Gaussian process is shown in the paper to follow a multivariate normal distribution. Analogously to the univariate case, if the regression parameters and the process variance are integrated out, the predictive distribution is multivariate t. Treating time as an input to the model, and building a standard single-output emulator based on this reformulation, is also considered. The multi-output emulator is found to perform better than the time-input emulator or several separate emulators in several examples. However, the computational cost of such an approach can be large, as many model runs may be required before the behaviour of the true model is properly understood.

Another alternative is to perform dimension reduction on the output. Instead of building an emulator for the simulator output directly, a set of basis functions are constructed which can be used to estimate the true output anywhere on a grid of points in time. This requires the number of basis functions to be determined, which can be done by considering the percentage of the variance explained by the approximation. Univariate Gaussian process emulators are then used to emulate the coefficients of the basis functions, and the simulator output at an untested point is predicted using a combination of the GP emulator and the basis function approximation. An early example of this method, in which much of the underlying theory is developed, is found in Higdon et al. (2008), where a principal component basis is used. Bowman and Woods (2016) take a similar approach but consider a thin-plate spline basis in addition to principal components; the thin-plate spline basis is found to perform better for dispersion modelling on a grid in time.

2.5 Conclusions

This chapter reviews the existing theory behind the process of approximating a complex computational model by means of a Gaussian process emulator. We reviewed possible correlation functions and regression components of the GP emulator, and presented posterior predictive distributions in a variety of cases from a partially Bayesian perspective. Methods to determine the unknown parameters of the emulator using either plug-in estimation or Markov chain Monte Carlo sampling were also reviewed, together with extensions of the GP emulator to properly handle a wider class of underlying computational models. Many of the ideas presented here will be taken forward in the context of chains of models in Chapter 3.

Chapter 3

Emulation for chains of multiple models

3.1 Introduction

A chain of multiple models is defined for the purposes of our work as a series of computational models with univariate output but potentially several input variables, in which the output of one model in the chain is used as an input variable to a second model and so on. We assume that the models are deterministic, so that the only uncertainty arises from the values of the input variables, and that the models are computationally expensive to run and must be approximated as described in Chapter 2.

There is a noteworthy body of literature on chains of computational models. The review paper of Stevens and Atamturktur (2016) considers several established approaches for verification and validation of predictions from a chain of models using real-life data. Examples of fields in which verification and validation processes for chains of models have been considered include anisotropic contact surfaces in material science by Konyukhov et al. (2008), and motion in a moored system in fluid dynamics by Lin and Yim (2006). In the context of Gaussian process emulation for chains of models, Damianou and Lawrence (2014) presented an approach to uncertainty propagation using variational inference. This approach expands on the work of Titsias and Lawrence (2010), and takes the output of the earlier model as an uncertain input to the next model, which is then treated as a latent variable. A prior distribution is then chosen for the latent space of the input, which allows it to be variationally integrated out using an approximation to its true posterior distribution.

The most relevant prior work is that of Kyzyurova et al. (2018), which developed methods to predict from a set of linked models in which the final model may take multiple inputs derived from distinct earlier models. This is a closely related problem to the one which forms the basis of this thesis, but is not identical in its formulation: it does not consider chains with more than two steps, while we do not allow more than one input per model to arise from earlier models. The related problem of Gaussian process emulation with an input known only up to a probability distribution is considered by Girard et al. (2002) and Candela et al. (2003), with a specific focus on time series forecasting. All of these papers are based on a theoretical result for the mean and variance of a linked emulator for a chain of two models, which we consider further in Section 3.3.

Before presenting our approach to the problem of emulating multiple models in a chain, we must define some notation. Let y_k be the output of the kth model in the chain, and let

$$\tilde{\mathbf{x}}_k = (x_{k,1}, ..., x_{k,qk})^T$$

be a vector of q_k distinct inputs to model k. We use this notation since (for example) $\mathbf{x}_1$ is potentially ambiguous given the existence of the inputs $x_{1,1}, x_{1,2}, ..., x_{2,1}, x_{3,1}, ...$. Each model in the chain does not necessarily takes the same number of inputs, so each q_k and thus the length of $\tilde{\mathbf{x}}_k$ may differ for distinct value of k.

For illustrative purposes, consider the simplest case: a chain of two models. The first model in the chain is defined by the equation

$$y_1 = \eta_1(\tilde{\mathbf{x}}_1)\,.$$

The second model in the chain is defined by the equation

$$y_2 = \eta_2(\tilde{\mathbf{x}}_2, y_1)\,,$$

as it depends not only on its own direct inputs $\tilde{\mathbf{x}}_2$ but also on the additional input y_1 which is the output of model 1. Such a chain is depicted in Figure 3.1.

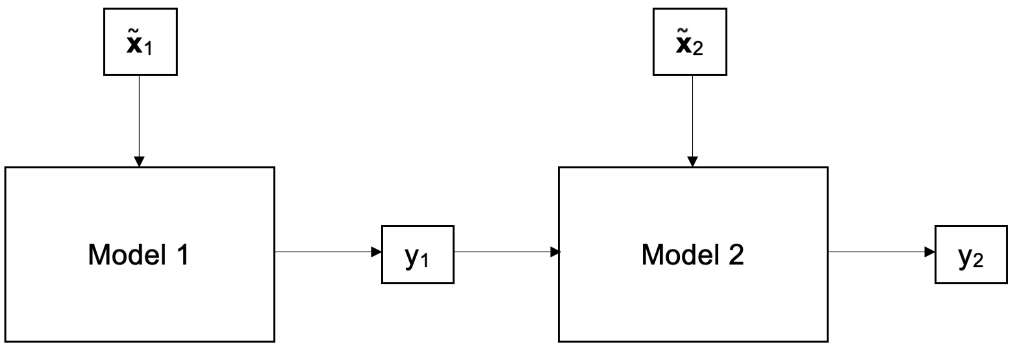

Figure 3.1: Diagram of a chain of two models

In order to make probabilistic predictions from multiple models in a chain, we need to understand how uncertainty in the models will combine. As discussed in Chapter 1, our work is based on emulating each model individually and linking the chain of emulators together, instead of building a composite emulator for the chain as a whole. The use of an emulator to approximate the first model in the chain introduces uncertainty in our predictions of the output, y_1, of model 1, which is then carried forward into the second model since y_1 is also an input to model 2. In longer chains, the same applies to the inputs y_2 to model 3, y_3 to model 4 and so on.

We define $\boldsymbol{\beta}_k$ as the vectors of regression coefficients for the emulator of model k, $\sigma^2_{z,k}$ as the prior variance for the emulator of model k, and $\boldsymbol{\theta}_k$ as the vectors of correlation parameters for the emulator for model k. The matrix of regression functions for the design points of the computer experiment for model k are denoted as $\mathbf{F}_k$, and the matrix of correlations between the design points as $\mathbf{C}_k$. The vector of observed outputs of the computer experiment at the n design points for model k is denoted by $\mathbf{Y}_{n,k}$.

We also require notation for prediction at an unknown set of inputs, analogous to the vector $\mathbf{x}_{n+1}$ in the single-model case. Let $\tilde{\mathbf{x}}_{k,n+1}$ be the vector of directly controllable inputs to model k at an untested input configuration, so that for a chain of r models, the set of all choosable inputs is $(\tilde{\mathbf{x}}_{1,n+1}, \tilde{\mathbf{x}}_{2,n+1}, ..., \tilde{\mathbf{x}}_{r,n+1})$. Similarly, let $\mathbf{x}_{k,n+1}$ be the vector of all inputs to the model k at an untested input configuration, including both the directly controllable inputs $\tilde{\mathbf{x}}_k$ and the unknown output y_{k-1} of the previous model in the chain. We define the vector of regression functions at $\mathbf{x}_{k,n+1}$ as $\mathbf{f}_{n+1,k}$, and the vector of correlations between $\mathbf{x}_{k,n+1}$ and the design points for model k as $\mathbf{c}_{n+1,k}$

We shall begin by simplifying the problem. Assume that weak priors are used throughout, and that the process variances and correlation parameters of every model in the chain are known. The emulator for model 1 is a standard GP emulator for a single model, as none of the inputs to the first model in the chain depend on the output of another model. Following equations (2.9) and (2.10), the posterior predictive distribution of y_1 is

$$f(Y_1|\tilde{\mathbf{x}}_1, \mathbf{Y}_{n,1}) \sim N(\mu_1, \sigma_1^2)\,,$$

where

$$\mu_1 = \mathbf{f}^T_{n+1,1}\hat{\boldsymbol{\beta}}_1 + \mathbf{c}^T_{n+1,1}\mathbf{C}_1^{-1}(\mathbf{y}_{n,1} - \mathbf{F}_1\hat{\boldsymbol{\beta}}_1)\,,$$

and

$$\sigma_1^2 = \sigma^2_{z,1}\left\{1 - (\mathbf{f}^T_{n+1,1}, \mathbf{c}^T_{n+1,1}) \begin{bmatrix} \mathbf{0} & \mathbf{F}_1^T \\ \mathbf{F}_1 & \mathbf{C}_1 \end{bmatrix} \begin{pmatrix} \mathbf{f}_{n+1,1} \\ \mathbf{c}_{n+1,1} \end{pmatrix} \right\}.$$

The estimate $\hat{\boldsymbol{\beta}}_1$ of the unknown regression coefficients $\boldsymbol{\beta}_1$ is the generalised least squares regression estimator,

$$\hat{\boldsymbol{\beta}}_1 = (\mathbf{F}_1^T\mathbf{C}_1^{-1}\mathbf{F}_1)^{-1}(\mathbf{F}_1^T\mathbf{C}_1^{-1}\mathbf{y}_{n,1})\,.$$

The emulator for a model $k > 1$ which occur later in the chain could still be treated as standard GP emulator if the input y_{k-1} is conditioned on. This would lead to posterior predictive distribution

$$f(Y_k|\tilde{\mathbf{x}}_k, y_1, \mathbf{Y}_{n,k}) \sim N(\mu_k, \sigma_k^2)\,,$$

where

$$\mu_k = \mathbf{f}_{n+1,k}^T\hat{\boldsymbol{\beta}}_k + \mathbf{c}_{n+1,k}^T\mathbf{C}_k^{-1}(\mathbf{y}_{n,k} - \mathbf{F}_k\hat{\boldsymbol{\beta}}_k)\,;$$

$$\sigma_k^2 = \sigma_{z,k}^2\left\{1 - (\mathbf{f}_{n+1,k}^T, \mathbf{c}_{n+1,k}^T)\begin{bmatrix}\mathbf{0} & \mathbf{F}_k^T\\ \mathbf{F}_k & \mathbf{C}_k\end{bmatrix}\begin{pmatrix}\mathbf{f}_{n+1,k}\\ \mathbf{c}_{n+1,k}\end{pmatrix}\right\};$$

$$\hat{\boldsymbol{\beta}}_k = (\mathbf{F}_k^T\mathbf{C}_k^{-1}\mathbf{F}_k)^{-1}(\mathbf{F}_k^T\mathbf{C}_k^{-1}\mathbf{y}_{n,k})\,.$$

For simplicity, we shall now consider a chain of two models only. The predictive distribution for y_2 depends on y_1. At an untested input configuration $\mathbf{x}_{1,n+1}$ of the inputs to the first model $\tilde{\mathbf{x}}_1$, the value of y_1 is known only up to its predictive distribution. The predictive distribution for y_2 thus depends on an uncertain input.

The ideal situation would be to integrate out y_1 altogether to obtain a predictive distribution which depends only on the directly controllable inputs to the chain of models:

$$f(y_2|\tilde{\mathbf{x}}_2, \tilde{\mathbf{x}}_1) = \int f(y_2|\tilde{\mathbf{x}}_2, y_1)f(y_1|\tilde{\mathbf{x}}_1)dy_1\,. \tag{3.1}$$

Let

$$f^*(y_2) = f(y_2|\tilde{\mathbf{x}}_2, y_1, \mathbf{Y}_{n,2})f(y_1|\tilde{\mathbf{x}}_1, \mathbf{Y}_{n,1})$$

be the product of the conditional densities for y_1 and y_2 given the outputs of the computer experiments for model 1 and model 2. This is given by

$$\begin{aligned} f^*(y_2) &= \frac{1}{\sqrt{2\pi\sigma_1^2}}\exp\left\{-\frac{(y_1-\mu_1)^2}{2\sigma_1^2}\right\} \times \frac{1}{\sqrt{2\pi\sigma_2^2}}\exp\left\{-\frac{(y_2-\mu_2)^2}{2\sigma_2^2}\right\}\\ &= \frac{1}{2\pi\sigma_1\sigma_2}\exp\left\{-\frac{(y_2-\mu_2)^2}{2\sigma_2^2} - \frac{(y_1-\mu_1)^2}{2\sigma_1^2}\right\}. \end{aligned} \tag{3.2}$$

In general, this is not integrable analytically, since μ_2 and σ_2^2 depend on y_1 in a complex, nonlinear way through $\mathbf{f}_{n+1,2}$ and $\mathbf{c}_{n+1,2}$:

$$\mathbf{f}_{n+1,2} = [f_{2,1}(x_{2,1}), f_{2,2}(x_{2,2}), ..., f_{2,q2}(x_{2,q2}), f_{2,q2+1}(y_1)]^T\,; \tag{3.3}$$

$$\mathbf{c}_{n+1,2} = [R(\mathbf{x}_{2,1} - \mathbf{x}_{2,n+1}), ..., R(\mathbf{x}_{2,n} - \mathbf{x}_{2,n+1})]^T\,. \tag{3.4}$$

A natural approach to consider so that the integral in (3.1) can be calculated exactly is to simplify the problem by constraining the form of $\mathbf{f}_{n+1,2}$ and $\mathbf{c}_{n+1,2}$. Assume a constant regression term such that every entry of $\mathbf{f}_{n+1,2}$ and $\mathbf{F}_2$ is equal to 1, and a squared exponential correlation function as defined in (2.2). Then,

$$\begin{aligned} R(\mathbf{x}_{2,1} - \mathbf{x}_{2,n+1}) &= \exp\Big\{ -\sum_{j=1}^{q_2} b_j(\mathbf{x}_{2,1}^{(j)} - \mathbf{x}_{2,n+1}^{(j)})^2 - b_{y1}(\mathbf{x}_{2,1}^{(q2+1)} - y_1)^2 \Big\} \\ &= \exp\Big\{ -\sum_{j=1}^{q_2} b_j(\mathbf{x}_{2,1}^{(j)} - \mathbf{x}_{2,n+1}^{(j)})^2 \Big\} \exp\Big\{ - b_{y1}(\mathbf{x}_{2,1}^{(q2+1)} - y_1)^2 \Big\}, \quad (3.5) \end{aligned}$$

where b_{y1} is the correlation parameter in the input corresponding to the output y_1 of the first model, and $\mathbf{x}_{2,1}^{(j)}$ is the jth entry of $\mathbf{x}_{2,1}$. Identical results are obtained for $R(\mathbf{x}_{2,2} - \mathbf{x}_{2,n+1}), ..., R(\mathbf{x}_{2,n} - \mathbf{x}_{2,n+1})$ up to different indexing. However, even after this simplification, it is still not possible to solve (3.1) analytically as the form of the integral is too complex. The remainder of this chapter considers two possible routes to make predictions from a chain of emulators despite this restriction.

3.2 Approximating the linked emulator output by Monte Carlo integration

Given the lack of an analytical solution to the integral of (3.2), a natural alternative is to approximate it using a numerical approach. We choose to use a simple form of Monte Carlo integration. For a given prediction point $(\tilde{\mathbf{x}}_{1,n+1}, \tilde{\mathbf{x}}_{2,n+1})^T$, we draw a sample of fixed size from the predictive distribution for y_1 given $\tilde{\mathbf{x}}_{1,n+1}$. We then draw a single value from the predictive distribution for y_2 given $\tilde{\mathbf{x}}_{2,n+1}$ and y_1 for each realisation of y_1. The generated values of y_2 then form an unbiased sample from the true posterior predictive distribution for y_2, which can be used to make inferences about the true value of y_2 at the prediction point.

In practise, the prior variances of the two Gaussian processes, $\sigma^2_{z,1}$ and $\sigma^2_{z,2}$, will not usually be known. A Bayesian approach is to condition on these quantities as well, leading to a different form for the integrand in (3.1). Following equations (2.4) and (2.7), the density that we wish to integrate with respect to y_1 is thus a product of two conditional t densities instead of normal densities. In the case of Monte Carlo integration, this does not change the approach described above. This approach still requires estimation of the correlation parameters of the GP emulators to the models in the chain, but could if desired be made fully Bayesian using MCMC.

As an example of the Monte Carlo strategy, we consider a chain of two models which are composed of known deterministic functions:

$$y_1(x_{1,1}, x_{1,2}) = \{\sin(2\pi x_{1,1}^3)\}^3\,, -0.8 \le x_{1,1} \le 0.8\,;$$

$$y_2(x_{2,1}, y_1) = \sin(\pi y_1) + \exp(y_1)\,. \tag{3.6}$$

Since this system is defined entirely by its first input, we can compare the true value of the target output, y_2, at a given value of $x_{1,1}$ to the predictions made by the emulator to test that the emulator's behaviour is reasonable.

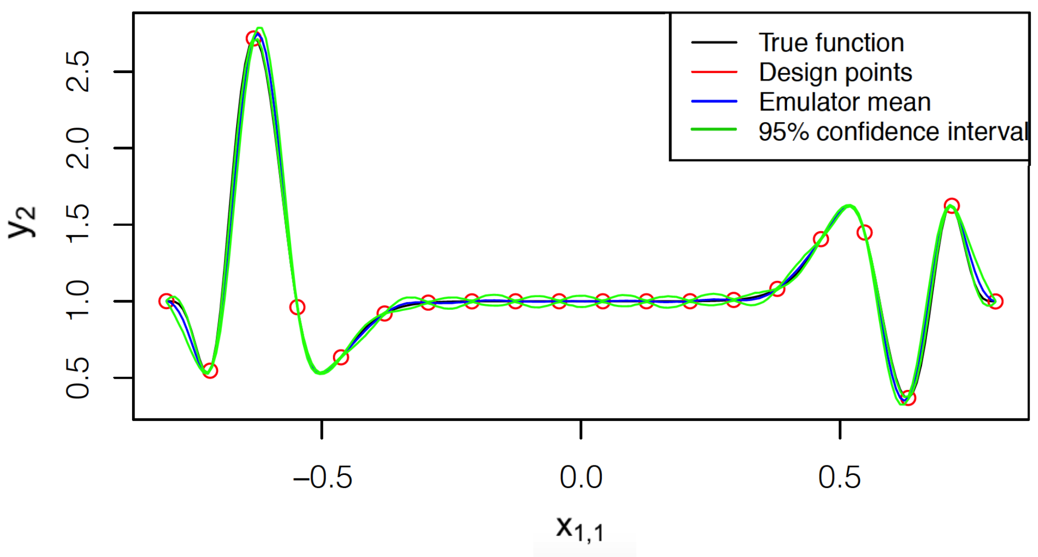

Figure 3.2: Prediction for y_2 against $x_{1,1}$ using the simulation method from the chain of emulators in the two-model example.

For our emulation, we use a Gaussian correlation function in both models, with a constant regression term and a nugget $\delta = 10^{-7}$. 20 design points are chosen, which are equally spaced in the critical input $x_{1,1}$ and randomly spaced in the trivial inputs $x_{1,2}$ and $x_{2,1}$. This would not be a feasible design for an unknown model, but is useful as an illustrative exercise to test that our predictions are reasonable. The outcome of interest is prediction from the posterior predictive distribution for y_2 across the range $-0.8 \leq x_{1,1} \leq 0.8$; the same ranges are used for $x_{1,2}$ and $x_{2,1}$, which assuming a reasonable correlation structure should have no impact on the predictions made.

The results of this are shown in figure 3.2. The true function is shown in black, the design points in red, the mean of the emulator at each $x_{1,1}$ in blue, and the upper and lower bounds of an empirical 95% prediction interval in green. These predictions tally well with what we would expect. The emulator mean tracks the true function well, and the uncertainty is at its largest away from the design points, pinching to virtually zero where the true output is known.

3.3 The mean and variance of the linked emulator

Although it is not possible to solve (3.1) analytically, which would produce a full distribution for the linked emulator, it is possible to obtain theoretical expressions for its mean and variance under certain conditions. This approach follows from the early work of Girard et al. (2002) and Candela et al. (2003) on Gaussian process emulation with uncertain inputs, and was developed in full for a chain of two models by Kyzyurova et al. (2018). Assume that all of the parameters of each emulator are obtained by a plug-in method, including the regression coefficients $\boldsymbol{\beta}_1$ and $\boldsymbol{\beta}_2$. The posterior predictive distributions on y_1 and y_2 given their GP emulators are found from equation (2.11); they are normal distributions, with means and variances

$$\mu_1 = \mathbf{f}_{n+1,1}^T\boldsymbol{\beta}_1 + \mathbf{c}_{n+1,1}^T\mathbf{C}_1^{-1}(\mathbf{y}_{n,1} - \mathbf{F}_1\boldsymbol{\beta}_1)\,;$$

$$\sigma_1^2 = \sigma_{z,1}^2(1 - \mathbf{c}_{n+1,1}^T\mathbf{C}_1\mathbf{c}_{n+1,1})\,;$$

$$\mu_2 = \mathbf{f}_{n+1,2}^T\boldsymbol{\beta}_2 + \mathbf{c}_{n+1,2}^T\mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \mathbf{F}_2\boldsymbol{\beta}_2)\,;$$

$$\sigma_2^2 = \sigma_{z,2}^2(1 - \mathbf{c}_{n+1,2}^T\mathbf{C}_2\mathbf{c}_{n+1,2})\,.$$

Assume a squared exponential correlation functions on the GP emulators for y_1 and y_2, and assume the regression term of the GP emulator is constant. Under these conditions, the following theorem holds for the output of a linked emulator for the model 2 output y_2:

Theorem 3.1. *Let*

$$E^*(y_2) = E(y_2|\tilde{\mathbf{x}}_2, \tilde{\mathbf{x}}_1, \mathbf{Y}_{n,2}, \mathbf{Y}_{n,1})$$

and let

$$Var^*(y_2) = Var(y_2|\tilde{\mathbf{x}}_2, \tilde{\mathbf{x}}_1, \mathbf{Y}_{n,2}, \mathbf{Y}_{n,1})$$

be the mean and variance of the posterior predictive distribution for y_2 under the linked emulator. Let $a^{(i)}$ be entry i of

$$\mathbf{a} = \mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \beta_{2,0})\,.$$

Then, the posterior predictive distribution for y_2 under the linked emulator has expectation

$$E^*(y_2) = \beta_{2,0} + \sum_{i=1}^{k} a^{(i)} \prod_{j=1}^{q_2} \exp\Big\{ - b_j(\mathbf{x}_{2,i}^{(j)} - \mathbf{x}_{2,n+1}^{(j)})^2\Big\} I_i \tag{3.7}$$

and variance

$$Var^*(y_2) = \sigma_{z,2}^2 - \Big[\sum_{i=1}^{k} a^{(i)} \prod_{j=1}^{q_2} \exp\{-b_j(\mathbf{x}_{2,i}^{(j)} - \mathbf{x}_{2,n+1}^{(j)})^2\} I_i\Big]^2$$
$$+ \sum_{i=1}^{k}\sum_{j=1}^{k}(a^{(i)}a^{(j)} - \sigma_{z,2}^2 C_2^{(i,j)}) \tag{3.8}$$
$$\prod_{d=1}^{q_2} \exp\{-b_d[(\mathbf{x}_{2,i,d} - \mathbf{x}_{2,n+1}^{(d)})^2 + (\mathbf{x}_{2,j}^{(d)} - \mathbf{x}_{2,n+1}^{(d)})^2]\} I_{i,j}\,,$$

where

$$I_i = \frac{1}{\sqrt{1+2\sigma_1^2 b_{y1}}} \exp\Big\{-\frac{(\mu_1 - \mathbf{x}_{2,i}^{(q_2+1)})^2}{\frac{1}{b_{y1}} + 2\sigma_1^2}\Big\}$$

and

$$I_{i,j} = \frac{1}{\sqrt{1+4\sigma_1^2 b_{y1}}} \exp\Big\{-\frac{(\mu_1 - \frac{\mathbf{x}_{2,i}^{(q_2+1)} + \mathbf{x}_{2,j}^{(q_2+1)}}{2})^2}{1/2b_{y1} + 2\sigma_1^2} - \frac{b_{y1}(\mathbf{x}_{2,i}^{(q_2+1)} - \mathbf{x}_{2,j}^{(q_2+1)})^2}{2}\Big\}.$$

Proof. Using the law of total expectation, the expectation of Y_2 can be expressed as

$$\begin{aligned} E^*(y_2) &= E\{E(y_2|\tilde{\mathbf{x}}_2, \mathbf{Y}_{n,2}, \mathbf{Y}_{n,1}, y_1)\} \\ &= E_{y1}(\mu_2) \\ &= E_{y1}[\beta_{2,0} + \mathbf{c}_{n+1,2}^T \mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \beta_{2,0})] \\ &= \beta_{2,0} + \int f(y_1)\mathbf{c}_{n+1,2}^T \mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \beta_{2,0}) dy_1 \\ &= \beta_{2,0} + \int \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\Big\{-\frac{(y_1-\mu_1)^2}{2\sigma_1^2}\Big\} \mathbf{c}_{n+1,2}^T \mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \beta_{2,0}) dy_1 \end{aligned} \tag{3.9}$$

where $\mathbf{c}_{n+1,2}$ is defined in (3.4). Let

$$\mathbf{a} = \mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \beta_{2,0})$$

and let $a^{(i)}$ and $c_2^{(i)}$ ($i = 1, ..., k$, where k is the number of design points to model 2) be the ith entry of $\mathbf{a}$ and $\mathbf{c}_{n+1,2}$ respectively. We may then write

$$\begin{aligned} \mathbf{c}_{n+1,2}^T \mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \beta_{2,0}) &= \mathbf{c}_{n+1,2}^T\,\mathbf{a} \\ &= \sum_{i=1}^{k} c_2^{(i)} a^{(i)}\,, \end{aligned}$$

which is equivalent to

$$\mathbf{c}_{n+1,2}^T\mathbf{C}_2^{-1}(\mathbf{y}_{n,2}-\beta_{2,0}) = \sum_{i=1}^{k} a^{(i)}\exp\{-b_{y1}(\mathbf{x}_{2,i}^{(q_2+1)}-y_1)^2\} \prod_{j=1}^{q_2}\exp\{-b_j(\mathbf{x}_{2,i}^{(j)}-\mathbf{x}_{2,n+1}^{(j)})^2\}\,.$$

This gives

$$E^*(y_2) = \beta_{2,0} + \int \frac{1}{\sqrt{2\pi\sigma_1^2}}\exp\left\{-\frac{(y_1-\mu_1)^2}{2\sigma_1^2}\right\} \sum_{i=1}^{k} a^{(i)}\exp\{-b_{y1}(\mathbf{x}_{2,i}^{(q_2+1)}-y_1)^2\}\prod_{j=1}^{q_2}\exp\{-b_j(\mathbf{x}_{2,i}^{(j)}-\mathbf{x}_{2,n+1}^{(j)})^2\}dy_1\,,$$

which may be written as

$$E^*(y_2) = \beta_{2,0} + \sum_{i=1}^{k} a^{(i)}\prod_{j=1}^{q_2}\exp\{-b_j(\mathbf{x}_{2,i}^{(j)}-\mathbf{x}_{2,n+1}^{(j)})^2\}I_i\,, \tag{3.10}$$

where

$$I_i = \frac{1}{\sqrt{2\pi\sigma_1^2}}\int \exp\left\{-\frac{(y_1^2-2y_1\mu_1+\mu_1^2)}{2\sigma_1^2} - b_{y1}([\mathbf{x}_{2,i}^{(q_2+1)}]^2 - 2\mathbf{x}_{2,i}^{(q_2+1)}y_1 + y_1^2)\right\}dy_1\,.$$

This is equivalent to

$$I_i = \frac{1}{\sqrt{2\pi\sigma_1^2}}\int \exp\left\{-\frac{(1+2\sigma_1^2 b_{y1})y_1^2 - 2y_1(\mu_1+2\sigma_1^2 b_{y1}\mathbf{x}_{2,i}^{(q_2+1)})}{2\sigma_1^2} - \frac{\mu_1^2+2\sigma_1^2 b_{y1}[\mathbf{x}_{2,i}^{(q_2+1)}]^2}{2\sigma_1^2}\right\}dy_1\,.$$

Completing the square and moving constants outside the integral gives

$$I_i = \frac{1}{\sqrt{1+2\sigma_1^2 b_{y1}}}\exp\left\{-\frac{\mu_1^2+2\sigma_1^2 b_{y1}[\mathbf{x}_{2,i}^{(q_2+1)}]^2 - \frac{(\mu_1+2\sigma_1^2 b_{y1}\mathbf{x}_{2,i}^{(q_2+1)})^2}{1+2\sigma_1^2 b_{y1}}}{2\sigma_1^2}\right\}\iota\,,$$

where

$$\iota = \int \frac{1}{\sqrt{2\pi\sigma_1^2/(1+2\sigma_1^2 b_{y1})}}\exp\left\{-\frac{(1+2\sigma_1^2 b_{y1})\left[y_1 - \frac{\mu_1+2\sigma_1^2 b_{y1}\mathbf{x}_{2,i}^{(q_2+1)}}{1+2\sigma_1^2 b_{y1}}\right]^2}{2\sigma_1^2}\right\}dy_1\,.$$

The integrand in this expression is the density of a normal distribution with mean $\frac{\mu_1+2\sigma_1^2 b_{y1}\mathbf{x}_{2,i}^{(q_2+1)}}{1+2\sigma_1^2 b_{y1}}$ and variance $\frac{\sigma_1^2}{1+2\sigma_1^2 b_{y1}}$. This integrates to 1, so $\iota = 1$ and therefore

$$
\begin{aligned}
I_i &= \frac{1}{\sqrt{1+2\sigma_1^2 b_{y1}}}\exp\left\{-\frac{\mu_1^2+2\sigma_1^2 b_{y1}[\mathbf{x}_{2,i}^{(q_2+1)}]^2-\frac{(\mu_1+2\sigma_1^2 b_{y1}\mathbf{x}_{2,i}^{(q_2+1)})^2}{1+2\sigma_1^2 b_{y1}}}{2\sigma_1^2}\right\}\\
&= \frac{1}{\sqrt{1+2\sigma_1^2 b_{y1}}}\exp\left\{-\frac{2\sigma_1^2 b_{y1}[\mathbf{x}_{2,i}^{(q_2+1)}]^2+2\mu_1^2\sigma_1^2 b_{y1}-4\mu_1\sigma_1^2 b_{y1}\mathbf{x}_{2,i}^{(q_2+1)}}{2\sigma_1^2(1+2\sigma_1^2 b_{y1})}\right\}\\
&= \frac{1}{\sqrt{1+2\sigma_1^2 b_{y1}}}\exp\left\{-\frac{[\mathbf{x}_{2,i}^{(q_2+1)}]^2+\mu_1^2-2\mu_1\mathbf{x}_{2,i}^{(q_2+1)}}{\frac{1}{b_{y1}}+2\sigma_1^2}\right\}\\
&= \frac{1}{\sqrt{1+2\sigma_1^2 b_{y1}}}\exp\left\{-\frac{(\mu_1-\mathbf{x}_{2,i}^{(q_2+1)})^2}{\frac{1}{b_{y1}}+2\sigma_1^2}\right\}, \qquad (3.11)
\end{aligned}
$$

which leads to the expression for $E^*(y_2)$ given in Theorem 3.1.

The variance of Y_2 can be expressed in terms of Y_1 using the law of total variance as

$$
\begin{aligned}
Var^*(y_2) &= Var(E[Y_2|\tilde{\mathbf{x}}_2,\mathbf{x}_1,\mathbf{Y}_{n,2},\mathbf{Y}_{n,1},y_1]) + E(Var[Y_2|\tilde{\mathbf{x}}_2,\tilde{\mathbf{x}}_1,\mathbf{Y}_{n,2},\mathbf{Y}_{n,1},y_1])\\
&= Var_{y1}(\mu_2)+E_{y1}(\sigma_2^2)\,. \qquad (3.12)
\end{aligned}
$$

The second term of (3.12) is

$$
\begin{aligned}
E_{y1}(\sigma_2^2) &= E_{y1}[\sigma_{z,2}^2(1-\mathbf{c}_{n+1,2}^T\mathbf{C}_2\mathbf{c}_{n+1,2})]\\
&= \sigma_{z,2}^2-\sigma_{z,2}^2\int f(y_1)\mathbf{c}_{n+1,2}^T\mathbf{C}_2\mathbf{c}_{n+1,2}dy_1\\
&= \sigma_{z,2}^2-\sigma_{z,2}^2\int\frac{1}{\sqrt{2\pi\sigma_1^2}}\exp\left\{-\frac{(y_1-\mu_1)^2}{2\sigma_1^2}\right\}\mathbf{c}_{n+1,2}^T\mathbf{C}_2\mathbf{c}_{n+1,2}dy_1 \qquad (3.13)
\end{aligned}
$$

We have

$$
\begin{aligned}
\mathbf{c}_{n+1,2}^T\mathbf{C}_2\mathbf{c}_{n+1,2} &= \sum_{i=1}^{k}(c_2^{(i)}\sum_{j=1}^{k}C_2^{(i,j)}c_2^{(j)})\\
&= \sum_{i=1}^{k}\sum_{j=1}^{k}c_2^{(i)}C_2^{(i,j)}c_2^{(j)}\,,
\end{aligned}
$$

where $C_2^{(i,j)}$ is the (i,j)th entry of $\mathbf{C}_2$. From the definition of $\mathbf{c}_{n+1,2}$ in equation (3.4) and the form of the Gaussian correlation function in equation (2.2), we can express $c_2^{(i)}$ and $c_2^{(j)}$ as the products of several exponential functions:

$$c_2^{(i)} = \exp\{-b_{y1}(\mathbf{x}_{2,i,p+1} - y_1)^2\} \prod_{d=1}^{q_2} \exp\{-b_d(\mathbf{x}_{2,i}^{(d)} - \mathbf{x}_{2,n+1}^{(d)})^2\}\,,$$

with an identical expression for $c_2^{(j)}$ except that every subscripted i is replaced with a j. Substituting this into (3.13) and grouping exponential terms gives

$$E_{y1}(\sigma_2^2) = \sigma_{z,2}^2 - \sigma_{z,2}^2 \sum_{i=1}^{k}\sum_{j=1}^{k} C_2^{(i,j)} \prod_{d=1}^{q_2} \exp\{-b_d[(\mathbf{x}_{2,i}^{(d)} - \mathbf{x}_{2,n+1}^{(d)})^2 + (\mathbf{x}_{2,j}^{(d)} - \mathbf{x}_{2,n+1}^{(d)})^2]\} I_{i,j}\,,$$

where

$$I_{i,j} = \frac{1}{\sqrt{2\pi\sigma_1^2}} \int \exp\left\{-\frac{(y_1-\mu_1)^2}{2\sigma_1^2}\right\} \exp\{-b_{y1}(\mathbf{x}_{2,i}^{(q_2+1)} - y_1)^2\} \exp\{-b_{y1}(\mathbf{x}_{2,j}^{(q_2+1)} - y_1)^2\} dy_1\,.$$

This can be rewritten as

$$I_{i,j} = \frac{1}{\sqrt{2\pi\sigma_1^2}} \int \exp\Bigg\{-\frac{(1+4\sigma_1^2 b_{y1})y_1^2 - 2y_1(\mu_1 + 2\sigma_1^2 b_{y1}\mathbf{x}_{2,i}^{(q_2+1)} + 2\sigma_1^2 b_{y1}\mathbf{x}_{2,j}^{(q_2+1)})}{2\sigma_1^2} + \frac{\mu_1^2 + 2\sigma_1^2 b_{y1}[\mathbf{x}_{2,i}^{(q_2+1)}]^2 + 2\sigma_1^2 b_{y1}[\mathbf{x}_{2,j}^{(q_2+1)}]^2}{2\sigma_1^2}\Bigg\} dy_1\,.$$

Completing the square and moving terms which do not depend on y_1 outside the integral leads to

$$I_{i,j} = \frac{1}{\sqrt{1+4\sigma_1^2 b_{y1}}} \exp\left\{-\frac{\mu_1^2 + 2\sigma_1^2 b_{y1}([\mathbf{x}_{2,i}^{(q_2+1)}]^2 + [\mathbf{x}_{2,j}^{(q_2+1)}]^2)}{2\sigma_1^2}\right\} \exp\left\{-\frac{(\mu_1 + 2\sigma_1^2 b_{y1}\mathbf{x}_{2,i}^{(q_2+1)} + 2\sigma_1^2 b_{y1}\mathbf{x}_{2,j}^{(q_2+1)})^2}{2\sigma_1^2(1+4\sigma_1^2 b_{y1})}\right\} \iota_{i,j}\,,$$

where

$$\iota_{i,j} = \int \frac{1}{\sqrt{2\pi\sigma_1^2/(1+4\sigma_1^2 b_{y1})}} \exp\left\{-\frac{(1+4\sigma_1^2 b_{y1})\left[y_1 - \frac{\mu_1 + 2\sigma_1^2 b_{y1}\mathbf{x}_{2,i}^{(q_2+1)} + 2\sigma_1^2 b_{y1}\mathbf{x}_{2,j}^{(q_2+1)}}{1+4\sigma_1^2 b_{y1}}\right]^2}{2\sigma_1^2}\right\} dy_1\,.$$

The integrand is the density of a normal distribution with mean $\frac{\mu_1 + 2\sigma_1^2 b_{y1}\mathbf{x}_{2,j}^{(q_2+1)} + 2\sigma_1^2 b_{y1}\mathbf{x}_{2,j}^{(q_2+1)}}{1+4\sigma_1^2 b_{y1}}$ and variance $\frac{\sigma_1^2}{1+4\sigma_1^2 b_{y1}}$, which integrates to 1 and therefore gives $\iota_{i,j} = 1$. The expression for $I_{i,j}$ therefore becomes

$$I_{i,j} = \frac{1}{\sqrt{1+4\sigma_1^2 b_{y1}}} \exp\left\{ -\frac{\mu_1^2 + 2\sigma_1^2 b_{y1}([\mathbf{x}_{2,i}^{(q_2+1)}]^2 + [\mathbf{x}_{2,j}^{(q_2+1)}]^2)}{2\sigma_1^2} \right.$$
$$\left. - \frac{(\mu_1 + 2\sigma_1^2 b_{y1}\mathbf{x}_{2,i}^{(q_2+1)} + 2\sigma_1^2 b_{y1}\mathbf{x}_{2,j}^{(q_2+1)})^2}{2\sigma_1^2(1+4\sigma_1^2 b_{y1})} \right\},$$

which can be rewritten and simplified to give

$$I_{i,j} = \frac{1}{\sqrt{1+4\sigma_1^2 b_{y1}}} \exp\left\{ -\frac{(\mu_1 - \frac{\mathbf{x}_{2,i}^{(q_2+1)}+\mathbf{x}_{2,j}^{(q_2+1)}}{2})^2}{1/(2b_{y1}) + 2\sigma_1^2} - \frac{b_{y1}(\mathbf{x}_{2,i}^{(q_2+1)} - \mathbf{x}_{2,j}^{(q_2+1)})^2}{2} \right\}. \quad (3.14)$$

Returning to equation (3.12), we must also find an expression for the first term. This can be rewritten as

$$Var_{y1}(\mu_2) = E_{y1}(\mu_2^2) - [E_{y1}(\mu_2)]^2$$

and since we have already determined $E_{y1}(\mu_2) = E^*(y_2)$ earlier in the theorem, only $E_{y1}(\mu_2^2)$ remains to be found. This can be written as

$$\begin{aligned} E_{y1}(\mu_2^2) &= E_{y1}\{[\beta_{2,0} + \mathbf{c}_{n+1,2}^T \mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \beta_{2,0})]^2\} \\ &= E_{y1}[\beta_{2,0}^2 + 2\beta_{2,0}\mathbf{c}_{n+1,2}^T \mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \beta_{2,0}) + \{\mathbf{c}_{n+1,2}^T \mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \beta_{2,0})\}^2] \\ &= \beta_{2,0}^2 + 2\beta_{2,0}E_{y1}[\mathbf{c}_{n+1,2}^T \mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \beta_{2,0})] + E_{y1}[\{\mathbf{c}_{n+1,2}^T \mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \beta_{2,0})\}^2]. \end{aligned} \quad (3.15)$$

An expression for $E_{y1}[\mathbf{c}_{n+1,2}^T \mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \beta_{2,0})]$ was derived in (3.9) and (3.10):

$$E_{y1}[\mathbf{c}_{n+1,2}^T \mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \beta_{2,0})] = \sum_{i=1}^{k} a^{(i)} \prod_{j=1}^{p} \exp\{-b_j(\mathbf{x}_{2,i}^{(j)} - \mathbf{x}_{2,n+1}^{(j)})^2\} I_i,$$

where I_i is given in (3.11). To deal with the final term of (3.15), we first note that

$$\begin{aligned} \{\mathbf{c}_{n+1,2}^T \mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \beta_{2,0})\}^2 &= (\mathbf{c}_{n+1,2}^T\, \mathbf{a})^2 \\ &= \sum_{i=1}^{k}\sum_{j=1}^{k} c_2^{(i)} c_2^{(j)} a^{(i)} a^{(j)}, \end{aligned}$$

so

$$E_{y1}[\{\mathbf{c}_{n+1,2}^T \mathbf{a})\}^2] = \int f(y_1) \sum_{i=1}^{k}\sum_{j=1}^{k} c_2^{(i)} c_2^{(j)} a^{(i)} a^{(j)}\, dy_1.$$

Expressing $c_2^{(i)}$ and $c_2^{(j)}$ as the products of several exponential functions as above, and collecting exponential and product terms, this is equal to

$$E_{y1}[\{\mathbf{c}_{n+1,2}^T\mathbf{a})\}^2] = \sum_{i=1}^{k}\sum_{j=1}^{k} a^{(i)}a^{(j)} \prod_{d=1}^{q_2} \exp\{-b_d[(\mathbf{x}_{2,i}^{(d)} - \mathbf{x}_{2,n+1}^{(d)})^2 + (\mathbf{x}_{2,j}^{(d)} - \mathbf{x}_{2,n+1}^{(d)})^2]I_{i,j} ,$$

where $I_{i,j}$ is defined in (3.14). Combining the terms of (3.12) calculated above, and substituting in the expression for $E^*(y_2)$ found in (3.10), gives the final result

$$\begin{aligned} Var^*(y_2) = \sigma_{z,2}^2 - &\left[\sum_{i=1}^{k} a^{(i)} \prod_{j=1}^{q_2} \exp\{-b_j(\mathbf{x}_{2,i}^{(j)} - \mathbf{x}_{2,n+1}^{(j)})^2\}I_i\right]^2 \\ &+ \sum_{i=1}^{k}\sum_{j=1}^{k}(a^{(i)}a^{(j)} - \sigma_{z,2}^2 C_2^{(i,j)}) \\ &\prod_{d=1}^{q_2} \exp\{-b_d[(\mathbf{x}_{2,i}^{(d)} - \mathbf{x}_{2,n+1}^{(d)})^2 + (\mathbf{x}_{2,j}^{(d)} - \mathbf{x}_{2,n+1}^{(d)})^2]\}I_{i,j} \end{aligned}$$

□

For an example of this approach, we consider a chain of two models each taking a single input. This example first appeared in Kyzyurova et al. (2018). The models are defined by the known deterministic functions

$$y_1 = 3x_{1,1} + \cos(5x_{1,1}) \, , -0.85 \le x_{1,1} \le 1 \, ,$$

and

$$y_2 = \cos(\frac{7y_1}{5}) - y_1 \, .$$

As in the original paper, six equally-spaced design points on $x_{1,1}$ are used to build the first emulator, with the output values of y_1 then taken as the design points for the second emulator. Prediction across the space of $x_{1,1}$ was then attempted using both the Monte Carlo method with a sample size of 10,000, and the theoretical approximation. The plot obtained from the theoretical approximation is shown in Figure 3.3; this is virtually identical to those obtained in the previous research. It is also useful to compare this plot to that obtained using the Monte Carlo method, which can be seen in Figure 3.4. This produces a plot which is almost indistinguishable from the theoretical method, with one exception: while the empirical variance bands are continuous and noiseless in Figure 3.3, the corresponding Monte Carlo result in Figure 3.4 has variance bands which are not completely smooth but instead have small but visible additional peaks and troughs, in particular when $x_{1,1}$ is between -1 and 0.5. This suggests that even a sample size of 10,000 is not sufficient to ensure that there is no noticeable Monte Carlo error in the results obtained.

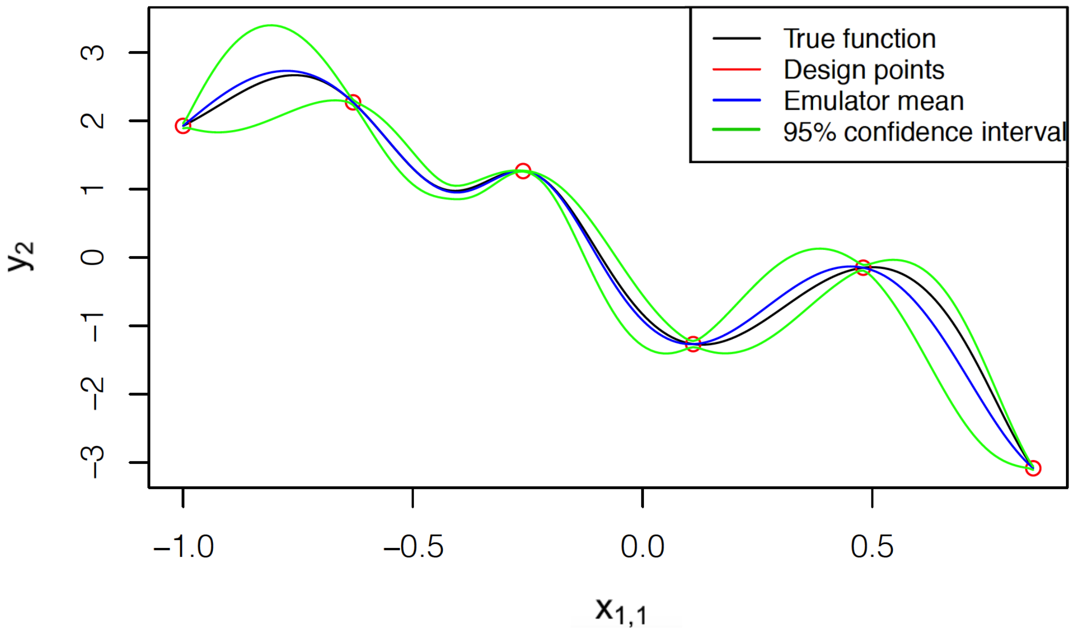

Figure 3.3: Prediction from the chain of emulators for y_2 against $x_{1,1}$ using the theoretical approximation.

3.4 Extending the simulation-based linked emulator to longer chains

To extend the simulation framework to chains of more than two models, we can apply the process described in Section 3.2 iteratively. For example, the posterior predictive distribution for the third model in the chain can be found from

$$f(y_3|\tilde{\mathbf{x}}_3, \tilde{\mathbf{x}}_2, \tilde{\mathbf{x}}_1) = \int\int f(y_3|\tilde{\mathbf{x}}_3, y_2) f(y_2|\tilde{\mathbf{x}}_2, y_1) f(y_1|\tilde{\mathbf{x}}_1) dy_2 dy_1 \,. \tag{3.16}$$

This result can be further generalised to chains of any length. The integrals can be done using a Monte Carlo method as in the two-model case with few changes: one implementational difference concerns the way in which variables must be stored within our code, since we now have multiple emulator outputs over which we must integrate.

We can test the Monte Carlo method on a chain of three closed-form deterministic functions which depend on a single input:

$$y_1(x_{1,1}) = x_{1,1} + sin(3\pi x_{1,1}) \,, 0 \leq x_{1,1} \leq 1 \,;$$

$$y_2(y_1) = y_1 - \log(1 + y1) \,;$$

$$y_3(y_2) = y_2 + \exp(-2y_2) \,. \tag{3.17}$$

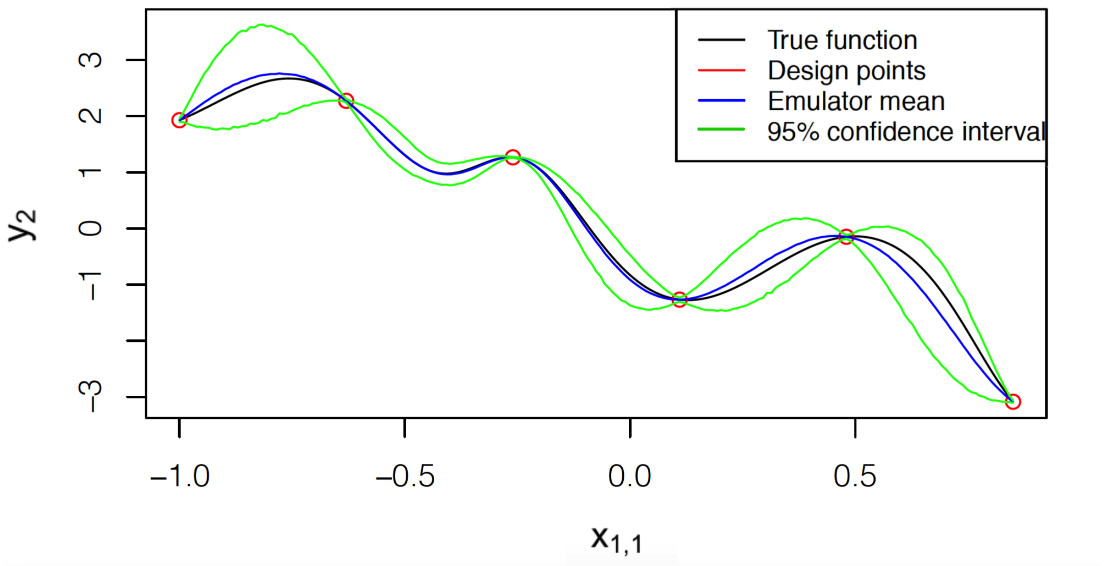

Figure 3.4: Prediction from the chain of emulators for y_2 against $x_{1,1}$ using the Monte Carlo method.

A Gaussian correlation and constant regression function are used in the three emulators, with nuggets of $\delta = 10^{-7}$ and ten design points spaced equally on $x_{1,1}$. The predictions from the chain for y_3 given $x_{1,1}$ are shown in Figure 3.5. Note that in this plot, the legend is omitted for clarity; the black, blue and green lines and the red circles have the same meaning as in Figure 3.4. The results appear reasonable, with largely accurate predictions across the input space, and uncertainty typically higher away from design points.

3.5 Extending the theoretical linked emulator to longer chains

Consider now a chain of three models in which Y_3 is a function of Y_2 and some $\tilde{\mathbf{x}}_3$, and Y_2 is a function of Y_1 and some $\tilde{\mathbf{x}}_2$. For ease of notation, let

$$E^*(y_3) = E(y_3|\tilde{\mathbf{x}}_3, \tilde{\mathbf{x}}_2, \tilde{\mathbf{x}}_1, \mathbf{Y}_{n,3}, \mathbf{Y}_{n,2}, \mathbf{Y}_{n,1})$$

be the posterior expectation of y_3 given the directly controllable inputs, observed data and the linked emulator. Applying the law of total expectation gives

$$\begin{aligned} E^*(y_3) &= E[E(y_3|\tilde{\mathbf{x}}_3, \mathbf{Y}_{n,3}, \mathbf{Y}_{n,2}, \mathbf{Y}_{n,1}, y_2)] \\ &= E_{y2}(\mu_3) \\ &= \int f(y_2)\mu_3 dy_2\,. \end{aligned} \tag{3.18}$$

This cannot be solved directly, since the full distribution $f(y_2)$ cannot be obtained in closed form. A seemingly natural alternative is to apply the law of total expectation

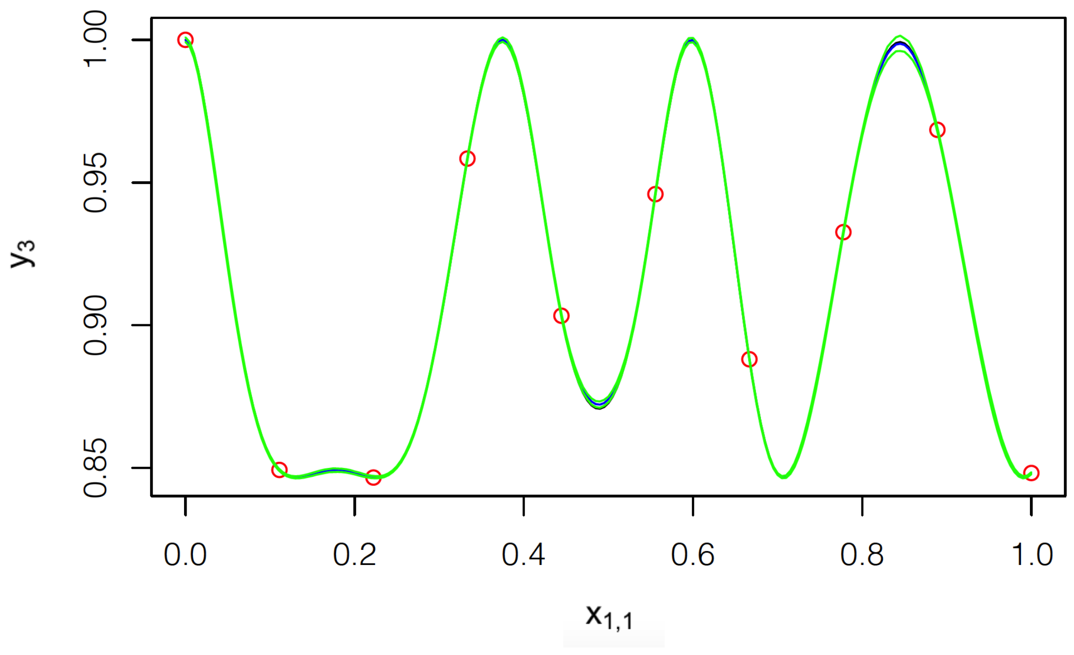

Figure 3.5: Prediction from the chain of emulators for y_3 against $x_{1,1}$ in the three-model example - simulation method.

again, giving

$$
\begin{aligned}
E^*(y_3) &= E_{y2}(\mu_3) \\
&= E_{y1}\{E_{y2}[\mu_3|y_1]\} \\
&= E_{y1}\{E_{y2}[\mu_3|y_1]\} \\
&= E_{y1}\{E_{y2}[(\beta_{3,0} + \mathbf{c}_{n+1,3}^T\mathbf{C}_3^{-1}(\mathbf{y}_{n,3} - \beta_{3,0})]\}
\end{aligned}
$$

where the final equality holds since, if y_1 is conditioned upon, y_2 follows a normal distribution with known mean and variance; this in turn implies that y_3 is the output of a GP emulator in which one input, y_2, is known only up to a normal distribution. The inner expectation is therefore identical to that in (3.9) with the model numbers within the chain increased by one, so its form is known from (3.7). Substituting this result gives

$$
\begin{aligned}
E^*(y_3) &= E_{y1}\left[\frac{1}{\sqrt{1+2\sigma_2^2 b_{y2}}}\exp\left\{-\frac{(\mu_2 - \mathbf{x}_{3,i}^{(q_3+1)})^2}{1/b_{y2}+2\sigma_2^2}\right\}\right] \\
&= \int f(y_1)\frac{1}{\sqrt{1+2\sigma_2^2 b_{y2}}}\exp\left\{-\frac{(\mu_2 - \mathbf{x}_{3,i}^{(q_3+1)})^2}{1/b_{y2}+2\sigma_2^2}\right\}dy_1\,. \qquad (3.19)
\end{aligned}
$$

This is still, however, analytically intractable - both the mean μ_2 and the variance σ_2^2 of the model 2 output y_2 depend on y_1 in complex ways. Their combined presence in three separate locations in the integrand of (3.19) means that the approach taken in the two-model case to reduce the integral to one which can be solved are not applicable here. Nor does this result naturally "scale up" to longer chains of models - even if (3.19) could be solved analytically, this would not mean that a four-model chain could be handled in the same way, since another application of the law of total expectation would be required. The same problems apply to calculating the variance of the linked emulator output, $Var^*(y_3)$, given the directly controllable inputs and the simulator runs.

A solution to this is provided by using an approximation to the distribution of y_2. When the theoretical results for the mean and variance of a GP emulator with an input known only up to a normal distribution were introduced in Girard et al. (2002), they were intended for use in time series analysis for multiple-step ahead forecasting - the same emulator would be used for each step in the chain. The problem of approximating the uncertain input at time steps later than the second was the reason for the use of the normal approximation on the linked emulator output. By Lemma 3.2, we can use this approach for predictions from chains of three or more models.

Lemma 3.2. *Let y_r be the output of a chain of r models, where the final model is defined by the equation*

$$y_r = \eta_r(\tilde{\mathbf{x}}_r, y_{r-1}) \, .$$

Let the distribution $f(y_{r-1})$ be approximated by a normal distribution with mean

$$\mu_{r-1} = \beta_{r-1,0} + \sum_{i=1}^{k} a^{(i)} \prod_{j=1}^{q_{r-1}} \exp\{-b_j(\mathbf{x}^{(j)}_{r-1,i} - \mathbf{x}^{(j)}_{r-1,n+1})^2\} I^{r-1}_i$$

and variance

$$\begin{aligned}\sigma^2_{r-1} = &\sigma^2_{z,r-1} - \left[\sum_{i=1}^{k} a^{(i)} \prod_{j=1}^{q_{r-1}} \exp\{-b_j(\mathbf{x}^{(j)}_{r-1,i} - \mathbf{x}^{(j)}_{r-1,n+1})^2\} I_i\right]^2 \\ &+ \sum_{i=1}^{k}\sum_{j=1}^{k}(a^{(i)}a^{(j)} - \sigma^2_{z,r-1}C^{(i,j)}_2) \\ &\prod_{d=1}^{q_{r-1}} \exp\{-b_d[(\mathbf{x}^{(d)}_{r-1,i} - \mathbf{x}^{(d)}_{r-1,n+1})^2 + (\mathbf{x}^{(d)}_{r-1,j} - \mathbf{x}^{(d)}_{r-1,n+1})^2]\} I^{r-1}_{i,j} \, ,\end{aligned}$$

where

$$I^{r-1}_i = \frac{1}{\sqrt{1 + 2\sigma^2_{r-2} b_{yr-1}}} \exp\left\{ - \frac{(\mu_{r-2} - \mathbf{x}^{(q_{r-1}+1)}_{r-1,i})^2}{1/b_{yr-2} + 2\sigma^2_{r-2}} \right\}$$

and

$$I_{i,j}^{r-1} = \frac{1}{\sqrt{1+4\sigma_{r-2}^2 b_{yr-2}}} \exp\left\{ - \frac{(\mu_{r-2} - \frac{\mathbf{x}_{r-1,i}^{(q_{r-1}+1)} + \mathbf{x}_{r-1,j}^{(q_{r-1}+1)}}{2})^2}{1/(2b_{yr-2}) + 2\sigma_{r-2}^2} - \frac{b_{yr-2}(\mathbf{x}_{r-1,i}^{(q_{r-1}+1)} - \mathbf{x}_{r-1,j}^{(q_{r-1}+1)})^2}{2} \right\}.$$

Then, y_r has expectation

$$E^*(y_r) = \beta_{r,0} + \sum_{i=1}^{k} a^{(i)} \prod_{j=1}^{q_2} \exp\{-b_j(\mathbf{x}_{r,i}^{(j)} - \mathbf{x}_{r,n+1}^{(j)})^2\} I_i^r$$

and variance

$$\begin{aligned} Var^*(y_r) = {} & \sigma_{z,r}^2 - \left[\sum_{i=1}^{k} a^{(i)} \prod_{j=1}^{q_r} \exp\{-b_j(\mathbf{x}_{r,i}^{(j)} - \mathbf{x}_{r,n+1}^{(j)})^2\} I_i \right]^2 \\ & + \sum_{i=1}^{k} \sum_{j=1}^{k} (a^{(i)} a^{(j)} - \sigma_{z,r}^2 C_2^{(i,j)}) \\ & \prod_{d=1}^{q_r} \exp\{-b_d[(\mathbf{x}_{r,i}^{(d)} - \mathbf{x}_{r,n+1}^{(d)})^2 + (\mathbf{x}_{r,j}^{(d)} - \mathbf{x}_{r,n+1}^{(d)})^2]\} I_{i,j}^r\,, \end{aligned}$$

where

$$I_i^r = \frac{1}{\sqrt{1+2\sigma_{r-1}^2 b_{yr-1}}} \exp\left\{ - \frac{(\mu_{r-1} - \mathbf{x}_{r,i}^{(q_r+1)})^2}{1/b_{yr-1} + 2\sigma_{r-1}^2} \right\}$$

and

$$I_{i,j}^r = \frac{1}{\sqrt{1+4\sigma_{r-2}^2 b_{yr-1}}} \exp\left\{ - \frac{(\mu_{r-1} - \frac{\mathbf{x}_{r,i}^{(q_r+1)} + \mathbf{x}_{r,j}^{(q_r+1)}}{2})^2}{1/(2b_{yr-1}) + 2\sigma_{r-1}^2} - \frac{b_{yr-1}(\mathbf{x}_{r,i}^{(q_r+1)} - \mathbf{x}_{r,j}^{(q_r+1)})^2}{2} \right\}.$$

Proof. Follows directly from the application of Theorem 3.1. □

It is informative to examine how this approach compares to the simulation method for the three-model example in Section 3.4. Using the same experimental design as before, we evaluate the chain as described above using a Normal approximation to y_2 given its theoretical mean and variance. The final output y_3 is plotted against $x_{1,1}$ in Figure 3.6. Again the legend is omitted for clarity; the black and blue lines and red circles have the same meaning as Figure 3.4. Here, however, the green lines for the error bounds are the mean plus or minus two standard deviations instead of an empirical 95% prediction interval.

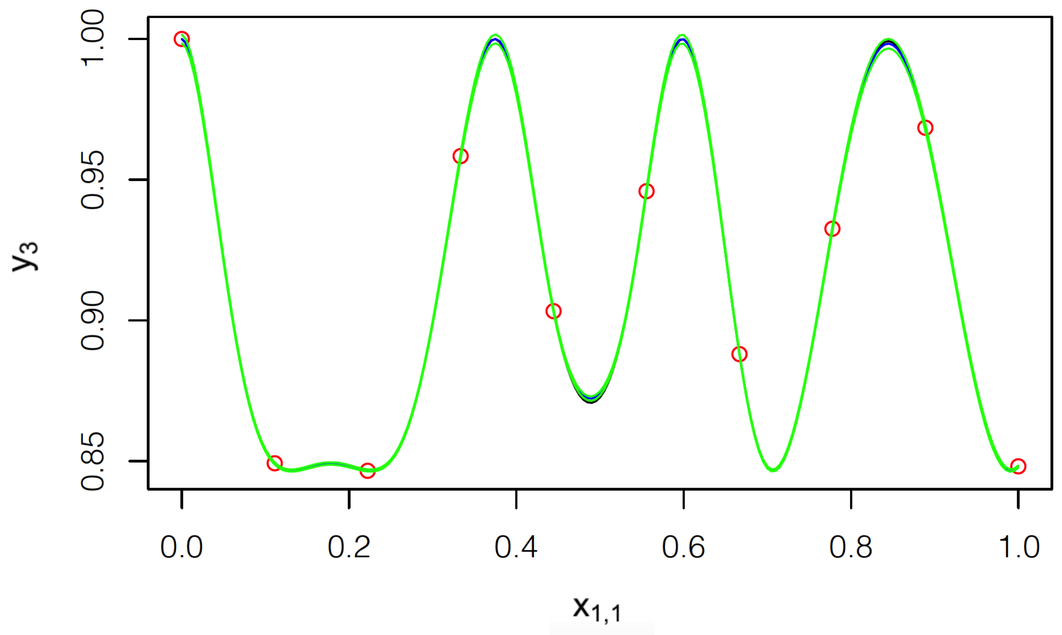

Figure 3.6: Prediction from the chain of emulators for y_3 against $x_{1,1}$ in the three-model example - theoretical method.

The results are similar to those seen in Figure 3.5. In particular, the mean of the linked emulator is extremely similar in both cases. There are however some noteworthy differences in the behaviour of the green lines. In general, we would expect that the prediction interval in the simulation method would be wider than the ± 2 s.d. bands in the theoretical method, since it takes account of the uncertainty in the process variances for each of the GP emulators in the chain by integrating them out; the theoretical method, by contrast, uses plug-in estimates for these parameters, which could have the effect of underestimating the true prediction variance. While this does occur in places, there are several locations in the prediction space where the bounds are in fact wider for the theoretical approach. This could be a result of the simulation method using a relatively small sample size of 1000 repetitions per prediction point, and the samples involved happening by chance to contain fewer extreme predictions than would be expected in general.

3.6 Conclusions

In this chapter, we have presented two methods to make predictions from a chain of computational models where emulation is required on the individual models in the chain. Neither approach is entirely new, but both are developed further here than in previous published work: earlier work on the simulation method has never been formalised to the extent presented in Section 3.2, while previous research on the theoretical method has been focused on the narrower problems of two-model chains or time series forecasting.

The methods introduced here form the basis of the remainder of this thesis. Chapters 4 and 5, dealing with experimental design and sensitivity analysis respectively, are both built around the requirements and capabilities of the theoretical and simulation-based linked emulators, which differ greatly from those of a single emulator in isolation. It is these methods which are implemented in code in Chapter 6, and which are demonstrated on a real problem in Chapter 7 and in a simulation study in Chapter 8. The alternative of a composite emulator for the entire chain is also considered in later chapters, but for reasons which will become clear when direct comparisons are available, we do not consider this to be a viable alternative in practice.

The two forms of linked emulator have very different properties. The theoretical method offers exact results for the expectation and variance of the posterior predictive distribution for the final model output under the linked emulator in a two-model chain, and near-exact results for longer chains - the normal approximation to the outputs of the intermediate models in Lemma 3.2 is the only reason why the results are not exact. It has the added benefit of being extremely fast to execute; this is discussed further in Chapter 6. Its main weakness is the highly restrictive set of assumptions under which it is valid. Only a Gaussian correlation function may be used, and plug-in estimation is required for a wide range of parameters of the individual emulators.

The simulation method offers more flexibility. It can be applied to emulators with any correlation function; different models in the chain could even use different correlation functions if desired. The variation arising from the process variance may be taken into account using a conjugate prior. The correlation parameters could in theory be handled using MCMC sampling, although this has not been implemented in our research. This form of linked emulator is however significantly more computationally intensive than the theoretical approach, since it requires a large number of runs from the chain of emulators. For a long chain with many inputs, very large Monte Carlo sample sizes may be required. The simulation method is also open to Monte Carlo error if the sample size chosen is too small, and it is not always possible to know in advance what sample size is appropriate for a given chain.

The differences between the two methods will be considered further in Chapter 8, where we shall demonstrate that even for short chains of relatively simple functions, the behaviour exhibited by the methods can vary greatly.

Chapter 4

Experimental design for chains of multiple models

4.1 Review of existing methods for experimental design

Experimental design for computer experiments concerns the choice of the design points at which the simulator is run to produce the training data. Choosing the design points effectively is especially important if the simulator is expensive or time-consuming to run, as it is then both impractical to run a large experiment and difficult to add additional design points if the result obtained is unsatisfactory. Intuitively, we would like the design points to cover the range of the input variables as well as possible, as for emulation our ability to predict at an unknown point is better the closer the point is to a design point. This approach is called space-filling design, and is widely used in previous literature. There are several techniques to achieve this goal.

The Latin hypercube design (LHD) was proposed by McKay et al. (1979). Given n design points, the design space χ is divided into n equally-sized intervals for each input variable , and the points are chosen such that exactly one point lies within each of these intervals. Such designs are very easy to create; Chapter 3 of Fang et al. (2006), for instance, described a simple algorithm to do so. Latin hypercube designs are popular for their good one-dimensional projection properties: Sacks et al. (1989) interprets the LHD as an extension of stratified sampling which ensures that the entire range of each input variable is covered by the design. However, this does not by itself guarantee that the points will cover the design space well; it is easy to generate an LHD which leaves areas of the design space poorly covered by the design points.

Another popular approach is to use an optimality criterion to generate a space-filling design. The two most common criteria were both proposed in Johnson et al. (1990) . Let ξ be any possible experimental design in χ, and let $d(\mathbf{x}_i, \mathbf{x}_j)$ be the distance between two points $\mathbf{x}_i$, $\mathbf{x}_j$. A maximin distance design maximises the criterion

$$\phi_{Mm}(\xi) = \min_{i \neq j} d(\mathbf{x}_i, \mathbf{x}_j)\,, \quad \mathbf{x}_i, \mathbf{x}_j \in \xi\,.$$

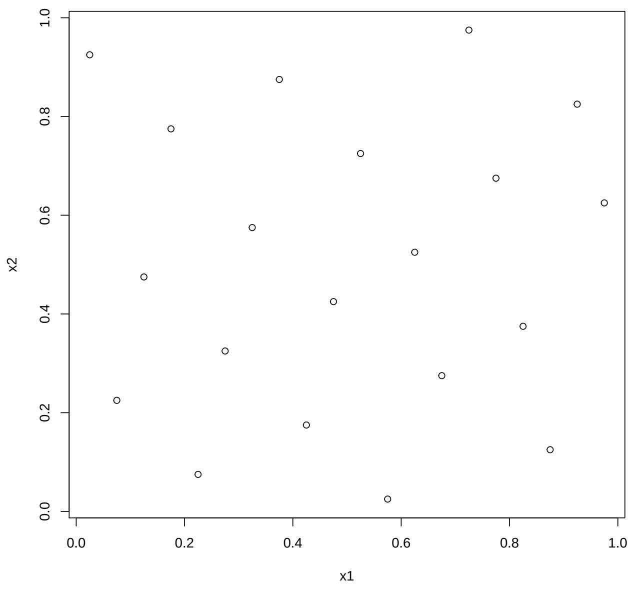

Figure 4.1: A maximin Latin hypercube design with 20 design points in two dimensions.

This is equivalent to maximising the minimum distance between any pair of design points, with the effect of spreading the design points as far across the design space as possible. This ensures that the design points do not cluster together.

A minimax distance design minimises the criterion

$$\phi_{mM}(\xi) = \max_{\mathbf{x}'} \min_{i=1,\dots,n} d(\mathbf{x}', \mathbf{x}_i)\,, \quad \mathbf{x}_i \in \xi, \quad \mathbf{x}' \in \chi\,.$$

In contrast to the maximin criterion, this minimises the maximum of the distance between every point in the design space and the nearest design point. This would appear useful for emulation, as an emulator can make better predictions at points close to a design point, so the minimax approach means that we can make reasonable predictions everywhere in the design space. However, minimax designs can be very computationally expensive to generate: instead of a single optimisation over a discrete region, we now require two optimisations, one of which is over a continuous space. For this reason they are less commonly used in practise.

An alternative is the coverage criterion, which is described in more detail by Nychka et al. (1997). A coverage design minimises the criterion

$$C_{k,l}(\xi) = \left\{ \sum_{\mathbf{x}' \in S} d_k(\mathbf{x}', \xi) \right\}^{(1/l)},$$

where

$$d_k(\mathbf{x}', \xi) = \left\{ \sum_{i=1}^{n} d(\mathbf{x}', \mathbf{x}_i)^k \right\}^{(1/k)},$$

k and l are tunable parameters and S is a set of candidate points in χ. The coverage criterion is much simpler to optimise than the minimax criterion, and can be made to approximate it: as $k \to -\infty$ and $l \to \infty$, $C_{k,l}(\xi) \to \phi_{mM}(\xi)$.

It is possible to combine the use of an optimality criterion and a Latin hypercube design: the class of designs which are considered is restricted to the set of possible LHDs, and the chosen criterion is considered only over this reduced set. The most frequently used example, primarily for reasons of ease of generation in practise, is the maximin LHD. Figure 4.1 shows an example of a 20-point maximin LHD in two dimensions.

The ease with which these forms of optimal design can be computed in practice varies widely between the specific designs. In general, coverage designs and Latin hypercube designs can be generated with relatively little computational effort and scale with increasing dimensionality. Maximin and especially minimax designs are significantly more computationally intensive. Nonetheless, there are pre-existing `R` packages which can generate all of these designs: coverage designs are supported by the `'fields'` package, presented by Nychka et al. (2016); maximin LHDs by the `'SLHD'` package, presented by Ba (2015); and maximin and minimax designs by the package `'minimaxdesign'`, presented by Mak and Joseph (2018).

The designs discussed above require that the number of design points is known in advance. It is possible that we would instead prefer to increment the number of design points after observing the results of an experiment for some initial set of points. This is called sequential or adaptive design, and was recommended for use in computer experiments in the pioneering paper of Sacks et al. (1989). The major advantage of such an approach is that the locations of the later design points can be chosen based on what is already known about both the true function and our initial emulator for it. For a large number of design points in many dimensions, sequential design may also be less computationally intensive than optimal single-stage design, since single-stage methods can suffer from the "curse of dimensionality" and become slow for large problems. Sequential design procedures may still make use of the space-filling designs described above, as a space-filling approach is a natural way to choose the initial set of design points.

The early work of Sacks et al. (1989) suggests three potential criteria by which the optimal location for a new design point may be determined, which remain the basis of many modern sequential design algorithms. Two are based on the Mean Square Prediction Error (MSPE). For a Gaussian Process emulator this is identical to the predictive variance, $\sigma^{*2}(\mathbf{x})$ (defined in Section 2.2), at an untested input point. The Maximum Mean Square Prediction Error (MMSPE) is defined as

$$\max_{\mathbf{x}\in\mathcal{X}} \sigma^{*2}(\mathbf{x}) \, ,$$

the maximum predictive variance of the GP emulator. The Integrated Mean Square Prediction Error (IMSPE) criterion is defined as

$$\int_{\mathcal{X}} \sigma^{*2}(\mathbf{x}) d\mathbf{x} \,,$$

the integral of the predictive variance across the design space χ. Both of these measures would be minimised by a good choice of design point. Also considered is a criterion based on maximising the expected entropy change as a result of a new design point, an idea which in the field of traditional statistical experimentation dates to Lindley (1956).

The MMSPE forms the basis of a method derived in MacKay (1992), and nowadays commonly known as Active Learning - MacKay (ALM). For an interpolating model to a true simulator, of which a GP emulator is an example, MacKay (1992) shows that - up to a quadratic approximation - the expected total gain in information about the simulator is maximised by placing a new design point at the location in the design space where the MSPE is highest. This supports the intuitive argument that determining the true value at the location where our knowledge about it is lowest is a good design principle. However, more recent authors including Beck and Guillas (2016) have noted that ALM's tendency to place many points on the boundaries of the experimental design region can be a weakness of the method, especially in high-dimensional problems.

An alternative algorithm proposed by Cohn et al. (1996), known as Active Learning - Cohn (ALC), places the new design point at location in the design space which minimises the expected IMSPE, thereby achieving the greatest expected reduction in the variance across the design space. For stationary Gaussian process emulators, Seo et al. (2000) compares the ALM and ALC algorithms, and concludes that ALC is generally preferable in terms of the fit of the emulator across the design space. The main drawback of ALC, as highlighted by Billonis and Zabaras (2012) and Jun and Horace (2009) amongst others, is its high computational cost.

Another approach to sequential design is based on the mutual information of two random variables, which was first used for experimental design by Caselton and Zidek (1984). The aim is to choose the design points such that the mutual information is maximised. For use in computer experiments, Krause et al. (2008) introduces an algorithm to maximise the mutual information sequentially; Beck and Guillas (2016) adapt this to deal with the case where a nugget is used in the Gaussian process, in the process introducing a new criterion named Mutual Information for Computer Experiments (MICE).

4.2 Single-stage design for chains of emulators

The design phase for multiple computer models faces additional challenges to that for a single model. When only one model is used, the space for each of the model inputs is known, and the value of the inputs at the design points can be directly chosen. Neither of these properties is true for variables such as y_1 and y_2, which - while acting as inputs to later models in the chain - depend on the values of the earlier inputs $x_{1,1}$, $x_{1,2}$, $x_{2,1}$ and so on. This leads to difficulties in adapting existing design methods to our new framework.

A natural first step in design for a chain of computer experiments is to use a space-filling design on the inputs to each model which we can choose directly, and use the outputs of the simulator runs for the earlier models as the design points for the inputs $y_1, y_2, \dots$ which come from earlier models in a chain. Since the space of y_1 is unknown, it appears intuitively sensible to use the output of y_1 at each of our design points for model 1 as our design points for y_1 in model 2. A first issue encountered with this approach is that, if the second simulator takes more than one input, the values of the other inputs at the design points must be chosen to avoid clashing with the pre-determined values of y_1. This can be done by generating a space-filling design on the new inputs, and permuting the values of y_1 to minimise the rank correlation with the other inputs, which ensures that the effect of y_1 is not confounded with that of other variables. This technique bears some resemblance to the work of Iman and Conover (1982), which considers inducing desired correlations onto the inputs of a computer model using rank correlation.

There is however a more significant issue with the principle of using the simulator runs from the first model as the design points for y_1 and so on. Space-filling design on a single model is largely used because it ensures that the input space is relatively well covered by the design points. When working with multiple models, however, space-filling on the inputs which we can choose directly does not guarantee a good spread of design points on the inputs we cannot directly control. In the simple two-model example in Section 3.2, the design points were chosen to be space-filling on the important input $x_{1,1}$ and random on the trivial input $x_{1,2}$, which ensured both that the GP emulator for the first model was able to identify that $x_{1,1}$ was a significant input and that we were able to make reasonable predictions across the space of $x_{1,1}$.

This does not, however, translate in general to a space-filling design for the second model. Consider a chain of three models defined by the functions

$$y_1(x_{1,1}) = x_{1,1} + \{\sin(2\pi x_{1,1}^3)\}^3 \, , -0.5 \le x_{1,1} \le 0.5 \, ;$$

$$y_2(y_1) = \{\sin(2\pi y_1^3)\}^3 \, ;$$

and

$$y_3(y_2) = \sin(\pi y_2) + \exp(y_2)\,.$$

We again use a Gaussian correlation function, constant regression term and nuggets $\delta = 10^{-7}$, and use the output of y_1 and y_2 as the design points for these variables when building emulators for the models to which they are inputs. 20 design points are chosen for $x_{1,1}$. These, and the corresponding values of y_1 and y_2 which go on to form the designs for the second and third models, are given in Table 4.1 to four decimal places.

$x_{1,1}$	y_1	y_2
-0.5	-0.8536	0.3328
-0.4474	-0.5991	-0.9296
-0.3947	-0.4483	-0.1542
-0.3421	-0.3575	-0.0227
-0.2895	-0.2930	-0.0039
-0.2368	-0.2374	-0.0006
-0.1842	-0.1843	0
-0.1316	-0.1316	0
-0.0789	-0.0789	0
-0.0263	-0.0263	0
0.0263	0.0263	0
0.0789	0.0789	0
0.1316	0.1316	0
0.1842	0.1843	0
0.2368	0.2374	0.0006
0.2895	0.2930	0.0039
0.3421	0.3575	0.0227
0.3947	0.4483	0.1542
0.4474	0.5991	0.9296
0.5	0.8536	-0.3328

Table 4.1: Initial experimental design for the three-model example.

Unlike in the examples in the previous chapter, this design is not satisfactory for this chain of models. As seen in Figure 4.2, while the predictions are good for much of the design space (with extremely low uncertainty over a large region), there is a noticeable problem at each extreme of the design space for $x_{1,1}$. The true function changes quickly in these regions and there are no more design points than in the rest of the space, with the result that our predictions are less accurate here. The estimate of the uncertainty is also lower than it should be, with a 95% prediction interval failing to include the true value in places.

In this chain, model 2 has large flat regions, and - as can be seen in Table 4.1 - many of the twenty design points lead to very similar values of $y_2 \approx 0$. This means that very little of the space of y_2 is covered by the design points. The result is the poor

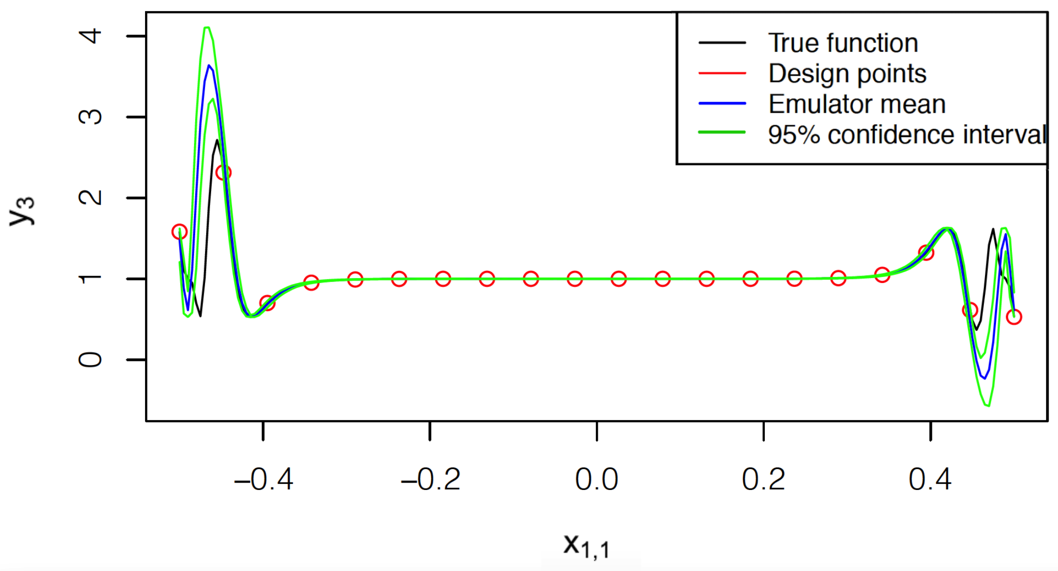

Figure 4.2: Prediction from the chain of emulators for y_3 against $x_{1,1}$ in the new three-model example, highlighting the experimental design issue.

predictions seen in Figure 4.2. Even greater issues could arise if more than one input existed to the chain as a whole.

In fact, the second function in this chain is so flat that this issue would arise even if the first model did not exist. Suppose the chain instead consisted of the true functions

$$y_1(x_{1,1}) = \{\sin(2\pi y_1^3)\}^3, -0.5 \leq x_{1,1} \leq 0.5\,;$$

$$y_2(y_1) = \sin(\pi y_1) + \exp(y_1)\,.$$

Figure 4.3 plots the values of y_1 arising from the 20 design points of $x_{1,1}$ given in Table 4.1, and the spacing of these points across the space of y_1, in this simpler case. Instead of 20 distinct design points, the design on y_1 for the second computer experiment is effectively based on only 11 points, and there are some large gaps between them. This demonstrates that the output of the simulator runs is not necessarily a good set of points to use for experimental design in future emulators.

Assuming it is possible to directly choose $y_1, y_2, \ldots, y_{n-1}$ at which we test the next model in the chain, a simple remedy for this problem exists. A new set of design points can be constructed for these inputs to later models. This allows $y_1, y_2, \ldots, y_{n-1}$ to be treated identically to the other inputs in terms of design, and means that existing space-filling design can be used on the later models without difficulty. The interpretation of the process is a little more complex, since the values of y_n obtained from the final computer experiment will not necessarily correspond to a specific set of inputs $\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \ldots$, but this does not hinder the creation of the linked emulator. The only problem remaining is to determine the range of the input. This can be done by using the

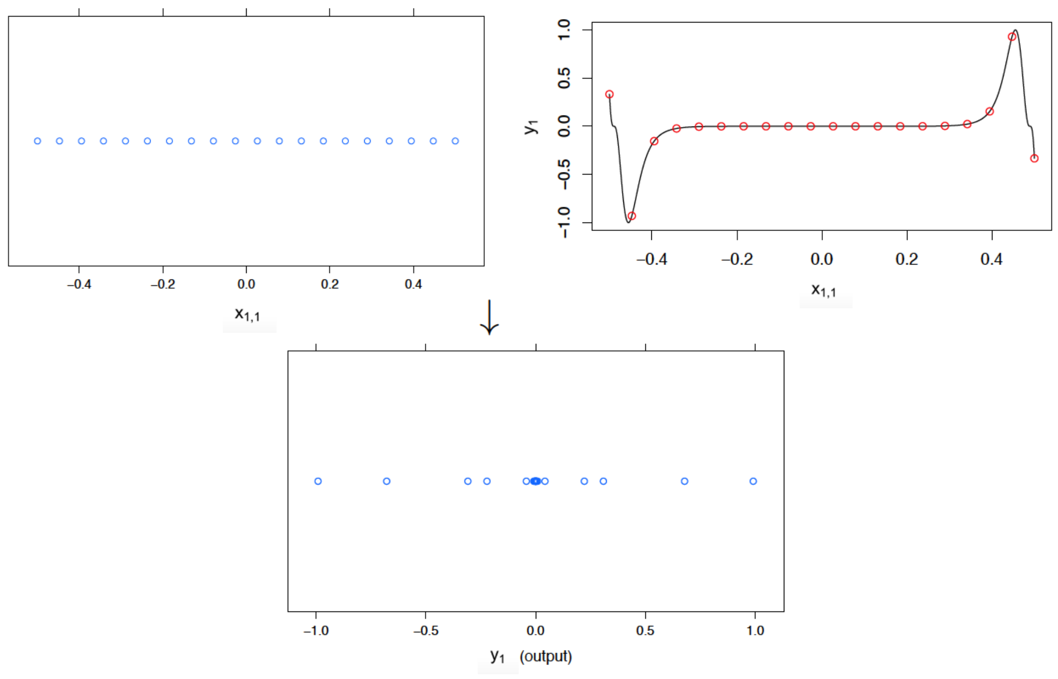

Figure 4.3: Example of a two-model chain in which space-filing on the input $x_{1,1}$ does not lead to a space-filling design on y_1.

largest and smallest values of the output for the previous model as an approximation to the boundaries of the variable, while constructing a new set of design points.

An additional benefit of this form of experimental design is that we are no longer forced to choose the same number of design points for each model in the chain. The models may have vastly different computational costs and numbers of inputs, and instead of fixing the number of design points d for every model, the available resources may thus be better used by allocating a different number of model runs, $n_1, n_2, ...n_r$ to each of the r models in the chain. A formal method for computer experimentation using this experimental design approach is given in Algorithm 2.

The stronger performance of this method relative to the naive approach of reusing the outputs can be seen by applying it to the three-model example presented above. For simplicity, we keep the number of design points for each model in the chain unchanged at 20. The new approach nonetheless leads to very different designs on y_1 and y_2. Together with the design points for $x_{1,1}$, these are given in Table 4.2.

The results, which can be seen in Figure 4.4, are also substantially different. The uncertainty in the predictions near the middle of the input space is larger, and there is new uncertainty at the $x_{1,1}$ design points arising from the design points in the later emulators no longer necessarily being at the values of y_1 and y_2 implied by the values of $x_{1,1}$ at which the first model was run. More importantly, the predictions at the fast-changing extremes of the true function have improved dramatically. Allowing the

Algorithm 2: Algorithm for computer experimentation using single-stage experimental design for a linked emulator

```
Input: Number of models, $r$; models in chain, $\eta_1, ..., \eta_r$; design for model 1, $\xi_1$;
       number of design points for each model, $n_1, ..., n_r$; number of inputs to
       each later model, $q_2, ..., q_r$; matrix of limits for controllable inputs to
       each later model, $\mathbf{L}_2, ..., \mathbf{L}_r$
Output: Experimental designs $\xi_1, ...\xi_r$ for each model in the chain; vector of
        outputs $\mathbf{y}_1, ..., \mathbf{y}_r$ at the design points identified for each model in
        the chain
begin
    $\mathbf{y}_1 \leftarrow$ vector of length $n_1$ ;
    for $i \leftarrow 1$ to $n_1$ do
        $\mathbf{y}_1^{(i)} \leftarrow \eta_1(\xi_1^{(i)})$ ;
        // $\mathbf{y}_1^{(i)}$ element $i$ of vector $\mathbf{y}_1$; $\xi_1^{(i)}$ row $i$ of design matrix $\xi_1$
    end
    for $k \leftarrow 2$ to $r$ do
        $\mathbf{L}_{y,k} \leftarrow$ vector $(\min(\mathbf{y}_{k-1}), \max(\mathbf{y}_{k-1}))$ ;
        $\xi_k \leftarrow$ design of size $n_k$ in $q_k$ dimensions scaled by relevant values of $\mathbf{L}_k$
          or $\mathbf{L}_{y,k}$ ;
        $\mathbf{y}_k \leftarrow$ vector of length $n_k$ ;
        for $j \leftarrow 1$ to $n_k$ do
            $\mathbf{y}_k^{(j)} \leftarrow \eta_j(\xi_k^{(j)})$ ;
        end
    end
end
```

whole space of y_1 and y_2 to be covered equally by design points means that the true shape of the function can be picked out more accurately by the chain of emulators, and with reduced uncertainty. This is largely because the behaviour of the third model at values of y_2 far away from 0 is better understood, leading to better predictions at the values of $x_{1,1}$ which give extreme values of y_2.

The method of experimentation presented in Algorithm 2 clearly offers many benefits when compared to naively using the output values as design points for later models. It is not, however, the final word on experimental design for chains of models. Single-stage design does not allow us to adapt our designs based on the results of the initial simulator runs. This is more relevant for a chain of models than for a single model, since the models may differ in several respects, including computational intensity and the size of the output variation with respect to their inputs. If this variation is understood in advance, single-stage design can take account of it, for example by allocating more runs to the less computationally intensive model. It is perhaps more likely that the variation will not be understood in advance. For this reason, sequential design is potentially more useful than single-stage design.

$x_{1,1}$	y_1	y_2
-0.5	-0.8536	-0.8559
-0.4474	-0.7637	-0.7658
-0.3947	-0.6739	-0.6757
-0.3421	-0.5840	-0.5856
-0.2895	-0.4942	-0.4955
-0.2368	-0.4043	-0.4054
-0.1842	-0.3145	-0.3153
-0.1316	-0.2246	-0.2252
-0.0789	-0.1348	-0.1351
-0.0263	-0.0449	-0.0450
0.0263	0.0449	0.0450
0.07894	0.1348	0.1351
0.1316	0.2246	0.2252
0.1842	0.3145	0.3153
0.2368	0.4043	0.4054
0.2895	0.4942	0.4955
0.3421	0.5840	0.5856
0.3947	0.6739	0.6757
0.4474	0.7637	0.7658
0.5	0.8536	0.8559

Table 4.2: Experimental design for the three-model example using Algorithm 2.

4.3 Sequential design for chains of emulators

There are several new challenges introduced to sequential design by the linking of multiple models. Firstly, methods for a single model rely on a closed-form expression of the variance of the emulator output at a given set of inputs, which is only available if the conditions imposed in Section 3.3 are applied here. The remainder of this section assumes that this is the case. A simple approach to sequential design for multiple models is to apply the ALM algorithm to the linked emulator: find the maximum value of the variance of the linked emulator, and add a design point at this location. The new design point returned will be a point in the space of all of the inputs to any model which do not depend on an earlier model. For example, in a chain of two models with inputs $\tilde{\mathbf{x}}_1^T$ and $(y_1, \tilde{\mathbf{x}}_2)^T$, the design point proposed will be a set of values for $\tilde{\mathbf{x}}_1$ and $\tilde{\mathbf{x}}_2$. This will therefore require both simulators to be run - the first at the set $\tilde{\mathbf{x}}_1^T$ proposed by the sequential design algorithm, and the second at the value of y_1 obtained from this run and the value of $\tilde{\mathbf{x}}_2$ found by the algorithm.

The problem can be visualised by considering the variance of the linked emulator in an example with two models. The first model takes a single input $x_{1,1}$ and returns output y_1. The second model takes two inputs, y_1 and $x_{2,1}$, and returns output y_2. We are interested in the variance of the emulator for y_2 given $(x_{1,1}, x_{2,1})^T$. We consider the system defined by the functions

$$y_1 = 3x_{1,1} + \cos(5x_{1,1}), -1 \leq x_{1,1} \leq 0.85\,;$$

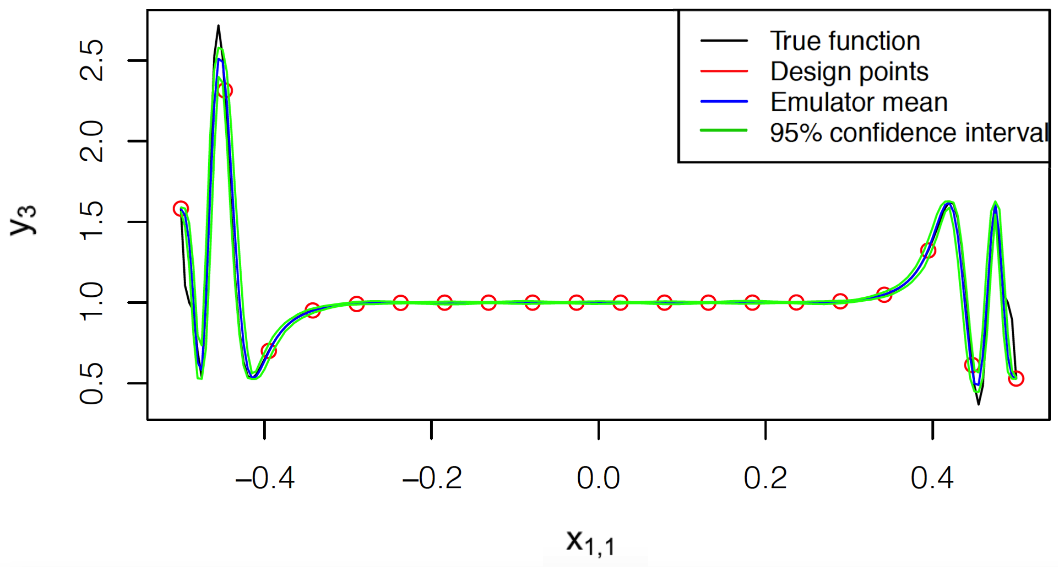

Figure 4.4: Prediction from the chain of emulators for y_3 against $x_{1,1}$ in the new three-model example with space-filling designs on each input. Note that here the red points are only the design points used for $x_{1,1}$; the resulting values of y_1 and y_2 may not correspond to design points for those variables.

$$y_2 = [\sin(2\pi y_1^3)]^3 + \sin(\pi x_{2,1}), -2.5 \leq x_{2,1} \leq 0.9\,.$$

The emulators use a Gaussian correlation function, with the regression coefficients and process variance estimated using maximum likelihood estimation and plugged in directly. This allows the theoretical method to obtain the variance of the linked emulator to be used.

For illustrative purposes, we consider two different initial design strategies. Six design points are chosen for model 1, using equal spacing on $x_{1,1}$. First, the outputs y_1 are used as the design points for the second model, combined with six equally spaced values of $x_{2,1}$ on the same range; of the 720 ways in which the two sets of values could be combined, the one with the least correlation between the two inputs is chosen. The variance of the linked emulator is plotted as a function of $x_{1,1}$ and $x_{2,1}$ in Figure 4.5, with the design points for $x_{1,1}$ (in the first computer experiment) and $x_{2,1}$ (in the second experiment) marked in black. This is similar to a standard plot of the variance of a single Gaussian process emulator in two dimensions, with a variance of zero at the design points, and much larger values as the distance from the design points increases. Away from the design points, parts of the variance surface is relatively flat, with many values close to the maximum value of 1.4695. This could make optimisation somewhat difficult, but suggests that many possible locations for a new design point would be of roughly equal value.

In the second instance, a maximin Latin hypercube design is used on for the second computer experiment, with the ranges of the two inputs taken from the output of the

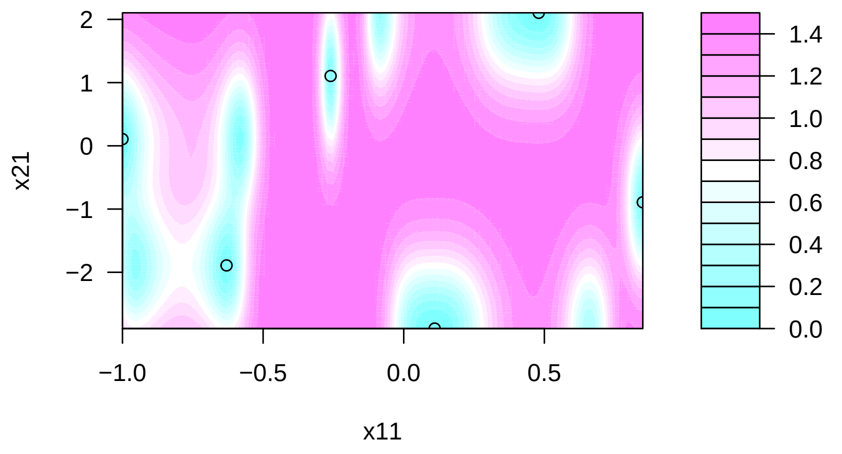

Figure 4.5: Variance of the linked emulator for y_2 given $x_{1,1}$ and $x_{2,1}$ under the first initial design.

simulator runs from model 1. The variance is given in Figure 4.6. No design points are marked here, since there is no unique pair of inputs $(x_{1,1}, x_{2,1})^T$ at which the full chain of models has been run; rather, we have an equally spaced set of values of $x_{1,1}$ at which the first model was run, and a set of values of $x_{2,1}$ at which the second model was run in tandem with values of y_1 which do not necessarily correspond to the values of y_1 obtained when the first model was run.

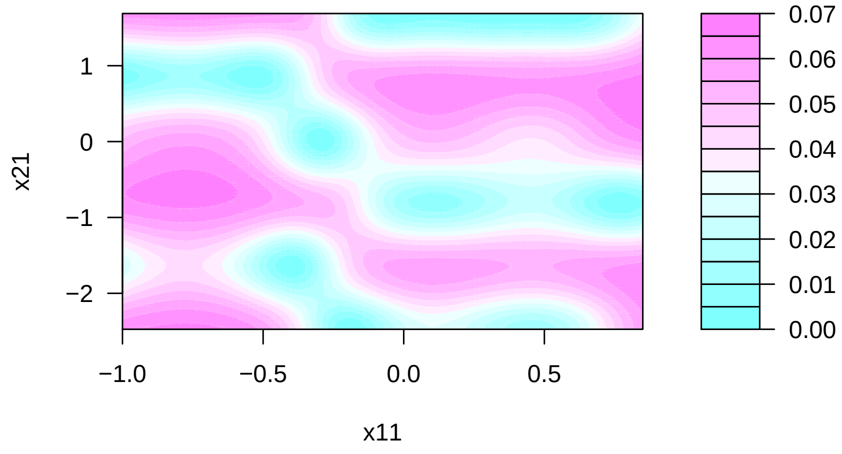

Figure 4.6: Variance of the linked emulator for y_2 given $x_{1,1}$ and $x_{2,1}$ under the second initial design.

In general, the variances are much lower than in the previous plot, with the largest observed variance now below 0.07 - less than 5 % of the maximum variance under the previous design. This is a result of the new design strategy, which allows for much more information about the behaviour of the second model to be obtained. However, this change also makes the plot more difficult to analyse, as the areas of lowest variance do not always correspond to the locations of design points in the space of $x_{1,1}$ and $x_{2,1}$ space. This is understandable: for example, a value of $x_{1,1}$ at which a design point was located may have led to a a value of y_1 which lies a relatively large distance from its nearest design points for the second experiment, so there may be relatively large uncertainty about the behaviour of the linked emulator at this point.

Finding the maximum predictive variance can be done relatively easily using the `"optim"` function in `R`, although care must be taken to ensure that the number starting points for the optimisation is large enough that the global optimum is found. Doing this for the example shown here results in a maximum of $x_{1,1} = 0.8500, x_{2,1} = 0.5468$ with a variance 0.0686. A simple sequential design method would now run the first simulator at $x_{1,1} = 0.8500$, store the result y_1^{new} obtained from this model run, and run the second model at the pair $(y_1^{new}, 0.5468)^T$. It is clear that a set of linked simulator runs at this point would have a beneficial effect on the quality of the linked emulator, since the variance of the emulator output would be reduced at the point where it is highest. In the following section, we discuss possible ways to improve upon this approach by taking account of the potential differences between the simulators in the chain.

4.4 Conclusions

In this chapter, we have reviewed methods for experimental design for computer experiments, and considered how they could be extended to deal with the additional challenges posed by the framework of a chain of computer models. We have discussed the reasons why using the output of earlier simulator runs as design points for later experiments is not in general a sensible strategy, and proposed an alternative with respect to single-stage designs. We also considered the more complex but potentially more rewarding case of sequential design for chains of computational models, and presented an example of what a simple approach to this could look like.

There are still significant weaknesses associated with the sequential design method considered above, however. If a very large proportion of the variance in the linked emulator at the point comes from variance in one emulator in the chain, the method may lead to resources being used for simulator runs which have little effect on the linked emulator. In the most extreme example, the value of y_1 may be known exactly at every point in $\tilde{\mathbf{x}}_1$, with all of the uncertainty in y_2 at any set of inputs arising from the second emulator. Unlikely though this case is, the principle it illustrates is likely to occur in practice for some chains: if one of the simulators is much more complex than the other and the number of initial design points is chosen to be the same for each model, the emulator of the more complex simulator is likely to have much greater uncertainty associated with its predictions.

One solution to this concern is to consider the variance in the individual emulators at the relevant inputs before running the associated model. The variance in the first emulator at the proposed set $\tilde{\mathbf{x}}_1$ would be evaluated first; if it was now decided not to run the model, the mean of the emulator output for y_1 at this point would be used as the value of y_1 for the next model in the chain. The same process would be performed for each model in the chain sequentially. This would in theory remove the unnecessary simulator runs, allowing the available budget to be used to greater effect elsewhere. The method to determine if the model should be run would reject a model run if the variance is below a certain threshold. Ideally, this should be chosen as a proportion of the total variance of the emulator, but it is unclear how this would be done for the earlier models in the chain, since there is no obvious general relationship between the variation in y_1 and the variation in y_2 when the model which links them is unknown away from its design points.

A related issue is that choosing the point in $\tilde{\mathbf{x}}_1$ which reduces the variance in y_2 the most does not guarantee that the implied value of y_1 has the same property. Running the first model to obtain an exact value of y_1 given $\tilde{\mathbf{x}}_1$ may remove much of the variance in y_2 immediately, and it is not clear that the obtained y_1 is necessarily the most effective value at which the second model can be run to achieve the maximum reduction of the variance of the linked emulator. It may instead be more beneficial to choose a new point in the design space of all of the inputs to model 2, $(\tilde{\mathbf{x}}_2, y_1)^T$, after running the first model.

Further complexities arise when the computational cost of the individual simulators is taken into account. If one model in the chain is ten times more expensive to run than another, for instance, a good method for sequential design would not assume that all of the models should be run equally often. For example, it may be more useful to run the cheaper model eleven times and the expensive model once than to run both models twice; the computational cost in both scenarios would be the same.

It may appear that simply looking at the individual emulators instead of the linked emulator would solve the problems, but this is not the case. This is because improvements to the separate emulators do not translate directly to the linked emulator. An extreme example occurs in a two-model chain in which y_1 is a complex function of $\tilde{\mathbf{x}}_1$, and y_2 is a function of y_1 and $\tilde{\mathbf{x}}_2$ in which y_1 plays little or no role in the value of y_2. If the second emulator is able to detect from its initial runs that y_1 is not an important input, then reducing the uncertainty in y_1 will have no effect on the linked emulator; the only way to improve the performance of the linked emulator is to add design points to the second emulator which improve our understanding of the effect of varying $\tilde{\mathbf{x}}_2$ on y_2. The individual emulators therefore cannot safely be used as an analogue for the linked emulator.

Chapter 5

Sensitivity analysis

5.1 Introduction

Sensitivity analysis is defined by Saltelli et al. (2008), Chapter 1, as “the study of how uncertainty in the output of a model can be apportioned to different sources of uncertainty in the model input”. This is distinct from uncertainty analysis, which seeks only to quantify the uncertainty instead of attribute it to its sources, although the two are often run in conjunction. Sensitivity analysis is useful for a number of reasons: it allows us to better understand the nature of a complex model, can help to reveal errors in the model formulation, establish which factors are most worthy of further research and may be used for model simplification. It is important to determine which of these is the main focus of a sensitivity analysis beforehand, as the failure to do so can lead to an unclear outcome of the analysis.

Our work is interested in extending the above definition of sensitivity analysis to encompass chains of models in addition to the single-model case. There are two related problems in the field sensitivity analysis for chains of linked models. The first concerns sensitivity analysis for the final model in a chain with respect to its own inputs, which is complicated by the fact that the distribution of the input y_{n-1} which arises from the output of an earlier model cannot be chosen directly. The second (and more complex) is that of sensitivity analysis for the output of the final model with respect to the directly controllable inputs to all of the model in the chain. Before considering these problems, however, it is necessary to review existing sensitivity analysis techniques for a single model.

Sensitivity analysis can be subdivided into local and global analysis. Local analysis uses the partial derivatives of the model to investigate the effect of small changes to the model inputs. Assuming the derivatives exist, this is computationally simple, but provides only limited insight into the effects of the inputs. The changes to the inputs that are of most interest are typically too large for local sensitivity analysis to capture. Global sensitivity analysis, which we shall focus on, provides much more information in this respect, but is less straightforward to perform. (Saltelli et al., 2008)

In its broadest sense, sensitivity analysis encompasses a very wide range of techniques. For instance, a simple visual representation of the effect of an input variable upon the model output can be obtained using a scatter plot of the input variable against the output variable. Performing this for multiple inputs allows us to qualitatively characterise the relative importance of the input variables (although care must be taken to account for changes in the other inputs, which may be difficult). This approach is used on many occasions by Saltelli et al. (2008) as a first step before attempting a more formal sensitivity analysis. Scatter plots however have only limited use for sensitivity analysis: they require a large amount of interpretation on the part of the researcher performing the analysis, provide no quantifiable measure of uncertainty and cannot account for the effects of interactions between the inputs.

More formal approaches towards sensitivity analysis aim to quantify the effect of an input (or set of inputs) numerically. There are still simple methods to do this: regression analysis, one of the most fundamental methods of applied statistics, can be viewed within this framework. Standardised regression coefficients provide a measure of the proportion of variability explained by each input. The problem with this, however, is that it depends on the relationship between the inputs and the output following predefined structure. If some of the variance explained by the inputs differs in scope from the assumed relationship, the regression coefficients may fail to give an accurate picture of the importance of each input. We would prefer our analysis to be independent of any assumed model.

Functional analysis of variance (FANOVA) extends traditional ANOVA methods to computational models. The strength of the relationship between the inputs and the output is quantified in terms of the effect of the input on the output variance. The effect of an input or set of inputs is defined by integrating out the other variables; under certain relatively relaxed conditions, this allows a simple decomposition of the output variance into the variance of the marginal effects of the inputs and their interactions (Schonlau and Welch, 2006). Probabilistic sensitivity analysis, which we use as the basis for our work, follows similar principals to FANOVA - although as we shall see, this comes with its own challenges.

5.2 Probabilistic sensitivity analysis

Probabilistic sensitivity analysis has been discussed by various authors including Oakley and O'Hagan (2004); the same approaches are covered by Saltelli et al. (2008), although the name is not used explicitly. First, we define some notation. As before, the inputs to a computational model consist of the vector $\mathbf{x} = (x_1, ..., x_q)^T$. Given a set of indices κ, the subvector of $\mathbf{x}$ containing the elements with these indices is written as $\mathbf{x}_\kappa$; $\mathbf{x}_{-\kappa}$ is a subvector of $\mathbf{x}$ containing all elements not in the set κ. In the applications we discuss here, κ is the empty set, a single index i or a pair of indexes i, j. The uncertainty in the input vector $\mathbf{x}$ is defined in terms of a probability distribution $G(\mathbf{x})$.

We treat the simulator output as a random variable Y with expectation $E(Y)$. The conditional expectation $E(Y|x_i)$ is the expectation of Y when the input x_i is fixed; for two fixed inputs x_i and x_j, the conditional expectation is $E(Y|\mathbf{x}_{i,j})$, and so on for larger sets of fixed inputs.

If we wish to understand how a model behaves with regards to its inputs, an important step is to decompose the model's output into a sum of main effects and interactions:

$$\eta(\mathbf{x}) = E(Y) + \sum_{i=1}^{q} z_i(x_i) + \sum_{i<j} z_{i,j}(\mathbf{x}_{i,j}) + ... + z_{1,2,...,d}(\mathbf{x}) \,,$$

where

$$z_i(x_i) = E(Y|x_i) - E(Y) \,, \tag{5.1}$$

and

$$z_{i,j}(\mathbf{x}_{i,j}) = E(Y|\mathbf{x}_{i,j}) - E(Y|x_i) - E(Y|x_j) + E(Y) \,,$$

with higher-order terms defined similarly. The value of these functions depend, as we would expect, on the choice of the distribution G. The function $z_i(x_i)$ is called the main effect of x_i; $z_{i,j}(\mathbf{x}_{i,j})$ is the second-order interaction between x_i and x_j. Higher-order interaction terms are similarly defined, although these terms are usually of less interest in sensitivity analysis. The main effects and second-order interactions, however, provide useful information on how the model output responds to the inputs. Welch et al. (1992) were an early example of authors considering this approach.

It can also be useful in probabilistic sensitivity analysis to characterise the sensitivity of the model output to its inputs in terms of the reduction in the variance of the output when some of its inputs are fixed. This approach is covered in detail by Chapter 4 of Saltelli et al. (2008). In general, for a subset of inputs κ, we define:

$$V_\kappa = var\{E(Y|\mathbf{X}_\kappa)\} \,, \tag{5.2}$$

where we treat the non-fixed inputs, denoted as $\mathbf{x}_{-\kappa}$, as a random variable $\mathbf{X}_{-\kappa}$. V_κ is based on the variance of the main effect of the subset of inputs $\mathbf{x}_\kappa$; it is the expected reduction in the variance of Y when $\mathbf{x}_\kappa$ is known. The measure can be normalised to be scale-invariant by dividing by the total variance $var(Y)$. The normalised index, S_κ, is usually referred to as a sensitivity index or Sobol' index.

For a single input, two special cases of equation (5.2) have been considered to quantify the reduction in variance associated with the input:

$$V_i = var\{E(Y|X_i)\} \,, \tag{5.3}$$

and

$$V_{Ti} = var(Y) - var\{E(Y|\mathbf{X}_{-i})\}\,,$$

which can be normalised to give the scale-invariant measures

$$S_i = \frac{var\{E(Y|X_i)\}}{var(Y)}\,,$$

and

$$\begin{aligned} S_{Ti} &= \frac{var(Y) - var\{E(Y|\mathbf{X}_{-i})\}}{var(Y)} \\ &= 1 - S_{-i}\,. \end{aligned}$$

The measure V_i is based on the variance of the main effect of x_i; it is the expected reduction in the variance of Y when x_i is known. In contrast, V_{Ti} is the remaining uncertainty when we know everything other than x_i. A larger value of V_i or V_{Ti} (or their scaled equivalents) indicates that the relevant x_i has a larger role in the uncertainty in Y. Dividing by the total variance $var(Y)$ transforms the absolute reduction in variance to a proportion, with the result that

$$\sum_{i=1}^{q} S_i + \sum_{i<j} S_{i,j} + \ldots + S_{1,2,\ldots,q} = 1\,,$$

since the variance in the output of a deterministic model is explained entirely by its inputs. The sum of the total effect indices S_{Ti} must be at least 1, and is equal to 1 only if the model is perfectly additive; if any interaction exists between any subset of the inputs, the sum will be greater than 1. It is also true that $S_{Ti} \geq S_i$ for any i.

When the aim is to establish which factors should be focused on in future with the aim of reducing uncertainty, Oakley and O'Hagan (2004) state that S_i provides the best measure of this; if precisely one factor could through some method be determined exactly, the factor with the largest S_i should be chosen. This does not necessarily extend to multiple factors, as the reduction in total variance then depends on the interaction between the factors as well as their main effects; the second-order interactions $S_{i,j}$ would also need to be considered.

It is possible to view model decomposition and variance-based sensitivity methods as complementary tools. Both exist within the same framework, and can thus convey similar information in different ways. Variance indicators provide a quantifiable measure of the proportion of the output variance explained by an input or set of inputs, which allows us to easily identify important inputs, but they cannot be used to determine how the model responds to these inputs. Model decomposition, by contrast, can be used for this: by plotting the expected output of the simulator given a fixed input at a range of values across the input's range, we can visualise the effect of the input on the output graphically. This can also be done for pairs of inputs using either three-dimensional plotting or contour plots, although it is more difficult for higher-order interactions. It can sometimes be difficult, however, to interpret these plots to determine which inputs or interactions are most important; this, together with dealing with higher-order interactions, is where the Sobol' indices are most powerful.

To actually compute either Sobol' indices or model decomposition measures requires several integrals to be calculated, since a number of expectations and variances are needed. For instance, to calculate V_i for an input i using equation 5.3, we need to calculate both the inner expectation (with respect to the other inputs $\mathbf{x}_{-i}$) and the outer variance (with respect to the input x_i). In general, these will not be available analytically, so numerical integration must be used. An adapted Monte Carlo integration method is typically chosen. Naively, this could be done as follows: a set of points is chosen from the marginal distribution on the input x_i, which we denote as $G_i(x_i)$, and another set of points are chosen from the conditional distribution of the other inputs $\mathbf{x}_{-i}$ given each x_i, which we denote as $G_{-i|i}(\mathbf{x}_{-i}|x_i)$. The inner expectation $E(Y|x_i)$ can then be estimated for each x_i, and the variance across these estimates calculated to give us the final indicator.

A weakness of this approach, however, is its computational feasibility. Accurate estimation of even a single indicator requires a large number of runs of the simulator at different input values. Saltelli et al. (2008), Chapter 4, provides a more efficient approach which substantially reduces the number of runs required, but even this is not always sufficient: O'Hagan (2006) describes a case in which variance-based sensitivity analysis requires several million runs, which is prohibitive even for a model of only moderate computational complexity.

For a complex model which cannot feasibly be run enough times to perform classical sensitivity analysis, several techniques are suggested by Saltelli et al. (2008), depending on the nature of the model. If the complexity in the model stems from an extremely large number of inputs, group sampling can be used: the inputs are grouped together into a relatively small number of blocks, and the effect of each block is investigated using the methods described above. This is far more computationally feasible, but can miss important inputs: if two variables in the same block have effects of similar size and opposite sign, their combined effect will be small, so the size of the effect of the block may not accurately reflect the importance of the inputs contained within it.

A more nuanced approach is the elementary effects (EE) method, introduced by Morris (1991) and refined by Campolongo et al. (2007). First, the space for each input is discretised to ρ levels. The elementary effect of the input x_i is defined as

$$EE_i = \frac{(Y|x_1,...x_{i-1},x_i+\Delta,x_{i+1},...,x_k) - (Y|x_1,...x_k)}{\Delta},$$

where Δ is a value in the set $\left[\frac{1}{\rho-1},\frac{2}{\rho-1},...,1-\frac{1}{\rho-1}\right]$; $\mathbf{x}$ must be chosen in such a way that the transformed point $\mathbf{x}+\Delta x_i$ is within the input space for every i. A measure of the sensitivity of the model output to each input can be obtained by calculating EE_i for each i at a set of l points across the input space and taking the mean of the absolute value of the elementary effects for i:

$$\mu_i^* = \frac{1}{l}\sum_{j=1}^{l}|EE_i^j|.$$

If an efficient sample of points is chosen, the number of model runs required in the EE method is far lower than in variance-based sensitivity analysis. Campolongo et al. (2007) demonstrated that μ_i^* is an effective proxy to S_{Ti}. Saltelli et al. (2008) recommend the use of the EE method for input screening for models where the number of inputs is sufficiently large that variance-based methods are impractical, but not so large that group sampling is required. Most authors are wary of using it as a method for sensitivity analysis in its own right, however: it can only measure the total effect of an input, not its main effect or the effect of specific interactions, and the discretisation step introduces some error into the calculation. It is usually performed as a first step to identify a subset of important inputs, which will then be the subject of a fuller sensitivity analysis.

While these methods may be very helpful when the number of inputs is very large, they cannot deal with the case of a model with a relatively small number of inputs which is extremely computationally intensive to run: neither group sampling nor screening using the EE method will be an effective method to reduce the number of simulator runs required. Emulation provides a possible solution.

5.3 Sensitivity analysis using emulation

Based on a small number of simulator runs, we can build a Gaussian Process emulator using the methods described in Chapter 2, with the correlation parameters estimated from the data. For the rest of this section, we assume that the regression coefficients and process variance are integrated out. This emulator can then be used to predict the simulator output at the much larger number of input configurations required to perform sensitivity analysis, overcoming many of the computational issues discussed above. This approach is presented in detail by Oakley and O'Hagan (2004).

Since a GP emulator provides as its output not a single value but a probability distribution for the true simulator output for a given set of inputs, the quantities of interest in sensitivity analysis also take the form of a distribution. We will denote expectations, variances and covariances with respect to the posterior distribution for the GP emulator by E^*, Var^* and Cov^* respectively. We denote by χ_κ the space of possible values for $\mathbf{x}_\kappa$, and by $\chi_{-\kappa}$ the space of possible values for $\mathbf{x}_{-\kappa}$. The marginal distribution of $\mathbf{x}_\kappa$ is called $G_\kappa(\mathbf{x}_\kappa)$; similarly, $G_{-\kappa|\kappa}(\mathbf{x}_{-\kappa}|\mathbf{x}_\kappa)$ is the conditional distribution of $\mathbf{x}_{-\kappa}$ given $\mathbf{x}_\kappa$.

The main effects and interactions of the input variables, which are linear functions of the GP emulator, a posteriori follow non-standardised t-distributions with $\nu_0 + n$ degrees of freedom (or n degrees of freedom when a non-informative prior is used). The means of these posterior distributions for the main effects and interactions can be found from a general result, given by Oakley and O'Hagan (2004):

$$E^*[E(Y|\mathbf{x}_\kappa)] = \mathbf{R}_\kappa(\mathbf{x}_\kappa)\hat{\boldsymbol{\beta}} + \mathbf{T}_\kappa(\mathbf{x}_\kappa)\mathbf{C}^{-1}(\mathbf{y}_n - \mathbf{F}\hat{\boldsymbol{\beta}})\,, \tag{5.4}$$

where $\hat{\boldsymbol{\beta}}$ is defined for strong prior distributions in (2.6) and for weak prior distributions in (2.8). $\mathbf{R}_\kappa(\mathbf{x}_\kappa)$ is defined as

$$\mathbf{R}_\kappa(\mathbf{x}_\kappa) = \int_{\chi_{-\kappa}} \mathbf{f}_{n+1}(\mathbf{x})\mathrm{d}G_{-\kappa|\kappa}(\mathbf{x}_{-\kappa}|\mathbf{x}_\kappa)\,, \tag{5.5}$$

the integral of the vector of regression functions at an unseen input configuration with respect to the distribution of the unknown inputs across the space of these inputs. Similarly, $\mathbf{T}_\kappa(\mathbf{x}_\kappa)$ is the integral of the vector of correlations between the unseen input configuration and the design points of the computer experiment with respect to the same distribution and space:

$$\mathbf{T}_\kappa(\mathbf{x}_\kappa) = \int_{\chi_{-\kappa}} \mathbf{c}_{n+1}(\mathbf{x})\mathrm{d}G_{-\kappa|\kappa}(\mathbf{x}_{-\kappa}|\mathbf{x}_\kappa)\,. \tag{5.6}$$

The fact that the vectors $\mathbf{f}_{n+1}$ and $\mathbf{c}_{n+1}$ are functions of the configuration of model input The special cases of (5.5) and (5.6) where κ is the empty set are denoted as $\mathbf{R}$ and $\mathbf{T}$ respectively, and are defined as

$$\mathbf{R} = \int_\chi \mathbf{f}_{n+1}(\mathbf{x})\mathrm{d}G\,,$$

and

$$\mathbf{T} = \int_\chi \mathbf{c}_{n+1}(\mathbf{x})\mathrm{d}G\,.$$

The posterior expectation of the function when no inputs are fixed then follows from (5.4) as

$$E^*[E(Y)] = \mathbf{R}\hat{\boldsymbol{\beta}} + \mathbf{T}\mathbf{C}^{-1}(\mathbf{y}_n - \mathbf{F}\hat{\boldsymbol{\beta}})\,. \tag{5.7}$$

The posterior mean of the main effect of an input x_i can be calculated as

$$E^*[z_i(x_i)] = [\mathbf{R}_i(x_i) - \mathbf{R}]\hat{\boldsymbol{\beta}} + [\mathbf{T}_i(x_i) - \mathbf{T}]\mathbf{C}^{-1}(\mathbf{y}_n - \mathbf{F}\hat{\boldsymbol{\beta}})\,, \tag{5.8}$$

where $z_i(x_i)$ is defined in (5.1). For the second-order interaction between inputs x_i and x_j, we obtain:

$$\begin{aligned} E^*[z_{i,j}(x_{i,j})] = {} & [\mathbf{R}_{i,j}(x_{i,j}) - \mathbf{R}_i(x_i) - \mathbf{R}_j(x_j) - \mathbf{R}]\hat{\boldsymbol{\beta}} \\ & + [\mathbf{T}_{i,j}(x_{i,j}) - \mathbf{T}_i(x_i) - \mathbf{T}_j(x_j) - \mathbf{T}]\mathbf{C}^{-1}(\mathbf{y}_n - \mathbf{F}\hat{\boldsymbol{\beta}})\,. \end{aligned} \tag{5.9}$$

A useful graphical summary of the effect of each input variable on the output can be obtained by plotting $E^*[z_i(x_i)]$ against x_i for $i = 1, ..., q$. It is tempting to assume that the input showing the most variation in this plot is the most important. However, Oakley and O'Hagan (2004) warn against this interpretation: the input showing the most variation has the largest value of $Var\{E^*[z_i(x_i)]\}$, but this is not the same as $E^*\{Var[z_i(x_i)]\}$. For this reason it can be useful to also consider the standard deviation of the posterior distribution for each input. Standard deviations of the t-distributions for the main effects and interactions can be calculated from another general result of Oakley and O'Hagan (2004). First, let

$$\begin{aligned} R^*(\mathbf{x}, \mathbf{x}') = {} & R(\mathbf{x} - \mathbf{x}') - \mathbf{c}_{n+1}(\mathbf{x})^T\mathbf{C}^{-1}\mathbf{c}_{n+1}(\mathbf{x}) + [\mathbf{f}_{n+1}(\mathbf{x})^T \\ & - \mathbf{c}_{n+1}(\mathbf{x})^T\mathbf{C}^{-1}\mathbf{F}](\mathbf{F}^T\mathbf{C}^{-1}\mathbf{F})^{-1}[\mathbf{f}_{n+1}(\mathbf{x}')^T - \mathbf{c}_{n+1}(\mathbf{x}')^T\mathbf{C}^{-1}\mathbf{F}]\,, \end{aligned}$$

This is sometimes referred to as the posterior correlation between $\mathbf{x}$ and $\mathbf{x}'$, and unlike the correlation function $R(\mathbf{x} - \mathbf{x}')$, it may depend on the locations of the points $\mathbf{x}$ and $\mathbf{x}'$ as well as the distance between them. Given the values of two different sets of fixed inputs, $\mathbf{x}_\kappa$ and $\mathbf{x'}_\upsilon$, let

$$U_{\kappa,\upsilon}(\mathbf{x}_\kappa, \mathbf{x'}_\upsilon) = \int_{\chi_{-\kappa}} \int_{\chi_{-\upsilon}} R^*(\mathbf{x}, \mathbf{x}')\mathrm{d}G_{-\kappa|\kappa}(\mathbf{x}_{-\kappa}|\mathbf{x}_\kappa)\mathrm{d}G_{-\upsilon|\upsilon}(\mathbf{x'}_{-\upsilon}|\mathbf{x'}_\upsilon)$$

be the integral of $R^*(\mathbf{x}, \mathbf{x}')$ with respect to the conditional distributions of the two sets of unknown inputs. The posterior covariance between the expectation of the simulator output given the two sets of fixed inputs is then

$$\begin{aligned} cov^*[E(Y|\mathbf{x}_\kappa), E(Y|\mathbf{x'}_\upsilon)] = {} & \hat{\sigma}^2[U_{\kappa,\upsilon}(\mathbf{x}_\kappa, \mathbf{x'}_\upsilon) - \mathbf{T}_\kappa(\mathbf{x}_\kappa)\mathbf{C}^{-1}\mathbf{T}_\upsilon(\mathbf{x'}_\upsilon)^T + \{\mathbf{R}_\kappa(\mathbf{x}_\kappa) \\ & - \mathbf{T}_\kappa(\mathbf{x}_\kappa)\mathbf{C}^{-1}\mathbf{F}\}(\mathbf{F}^T\mathbf{C}^{-1}\mathbf{F})^{-1}\{\mathbf{R}_\upsilon(\mathbf{x'}_\upsilon) - \mathbf{T}_\upsilon(\mathbf{x'}_\upsilon)\mathbf{C}^{-1}\mathbf{F}\}^T]\,, \end{aligned} \tag{5.10}$$

where the estimate $\hat{\sigma}^2$ of the unknown process variance σ^2 is

$$\hat{\sigma}^2 = \frac{c_0 + \mathbf{b}_0^T\mathbf{V}_0^{-1}\mathbf{b}_0 + \mathbf{y}_n^T\mathbf{C}^{-1}\mathbf{y}_n - \hat{\boldsymbol{\beta}}^T(\mathbf{V}_0^{-1} + \mathbf{F}^T\mathbf{C}^{-1}\mathbf{F})^{-1}\hat{\boldsymbol{\beta}}}{n + \nu_0 - 2}\,, \tag{5.11}$$

and $\hat{\boldsymbol{\beta}}$ is defined in equation (2.6). Equation (5.11) reduces when weak prior distributions are used to

$$\hat{\sigma}^2 = \frac{\mathbf{y}_n^T \mathbf{C}^{-1} \mathbf{y}_n - \hat{\boldsymbol{\beta}}^T (\mathbf{F}^T \mathbf{C}^{-1} \mathbf{F})^{-1} \hat{\boldsymbol{\beta}}}{n-2},$$

where $\hat{\boldsymbol{\beta}}$ takes the simpler form seen in equation (2.8).

In practice, we would generally prefer to plot $E^*[E(Y|x_i)]$ (a simple rescaling of $E^*[z_i(x_i)]$ in which the mean is not subtracted) to see how the fixed input changes the expected output in absolute terms instead of in isolation. Similarly, a two-dimensional contour or level plot can be used to display the shape of the function $E^*[E(Y|x_i, x_j)]$ and thus investigate how changing two factors in tandem affects the expected output.

The Sobol' index S_i and the total effect index S_{Ti} are quadratic functions of the GP emulator, so their posterior distributions are not t-distributions; indeed, they cannot be derived analytically. We can, however, obtain the posterior means of the unnormalised quantities V_i and V_{Ti}. In theory, their posterior variances could also be calculated, but the formulae required to do this are extremely complicated and are omitted by existing literature on the topic such as the important papers of Haylock and O'Hagan (1996) and Oakley and O'Hagan (2004).

The posterior means of V_i and V_{Ti}, which we denote by $E^*(V_i)$ and $E^*(V_{Ti})$, can be found from a result - again given by Oakley and O'Hagan (2004) - for the more general case of $E^*(V_\kappa) = E^*\{Var[E(Y|\mathbf{x}_\kappa)]\}$. We note as an intermediate step that

$$\begin{aligned} V_\kappa &= E[E(Y|\mathbf{x}_\kappa)^2] - E[E(Y|\mathbf{x}_\kappa)]^2 \\ &= E[E(Y|\mathbf{x}_\kappa)^2] - E(Y)^2, \end{aligned}$$

and that we can already calculate $E^*[E(Y)^2]$ from the results given in equations (5.7) and (5.10), since

$$E^*[E(Y)^2] = \{E^*[E(Y)]\}^2 + Var^*[E(Y)]. \tag{5.12}$$

Thus $E^*(V_\kappa)$ can be calculated if we can find an expression for $E^*\{E[E(Y|\mathbf{x}_\kappa)^2]\}$. This is given by

$$\begin{aligned} E^*\{E[E(Y|\mathbf{x}_\kappa)^2]\} = \int_{\chi_\kappa} \int_{\chi_{-\kappa}} \int_{\chi_{-\kappa}} & [\hat{\sigma}^2 R^*(\mathbf{x}, \mathbf{x}^*) + \mu^*(\mathbf{x})\mu^*(\mathbf{x}^*)] \\ & \mathrm{d}G_{-\kappa|\kappa}(\mathbf{x}_{-\kappa}|\mathbf{x}_\kappa)\mathrm{d}G_{-\kappa|\kappa}(\mathbf{x'}_{-\kappa}|\mathbf{x'}_\kappa)\mathrm{d}G_\kappa(\mathbf{x}_\kappa), \end{aligned} \tag{5.13}$$

where $\mathbf{x}^*$ is a vector containing the elements of $\mathbf{x}_\kappa$ and $\mathbf{x'}_{-\kappa}$ (this is not usually equal to $\mathbf{x}$, as $\mathbf{x}$ and $\mathbf{x}'$ are distinct), and μ^* is given in equation (2.4) for strong prior distributions and in equation (2.7) for weak priors. Where previously μ^* was defined implicitly in terms of $\mathbf{x}_{n+1}$, in this chapter, we treat μ^* explicitly as a function, as we are interested in the posterior mean at multiple untested input configurations.

Having obtained estimates for $E^*(V_\kappa)$ and $E^*(V_{T\kappa})$, we can divide them by $E^*[var(Y)]$ for inference about S_κ and $S_{T\kappa}$ (although it should be noted that this does not give $E^*(S_\kappa)$ and $E^*(S_{T\kappa})$, which cannot be derived analytically). To obtain $E^*[var(Y)]$, we use a similar approach to that used to obtain $E^*(V_\kappa)$:

$$E^*[Var(Y)] = E^*[E(Y^2)] - E^*[E(Y)^2]\,.$$

Equation (5.12) gives us an expression for the second term, so again we need only the first, which can be found from a result from Haylock and O'Hagan (1996):

$$\begin{aligned} E^*[E(Y^2)] &= \int_\chi E^*[\eta^2(\mathbf{x})]\mathrm{d}G(\mathbf{x}) \\ &= \int_\chi [\hat{\sigma}^2 R^*(\mathbf{x},\mathbf{x}) + \mu^*(\mathbf{x})^2]\mathrm{d}G(\mathbf{x})\,. \end{aligned} \tag{5.14}$$

5.4 Practical issues

There are several practical considerations relating to the calculation of the indices described above. An in-depth discussion of several of these can be found in Le Gratiet et al. (2014). This paper recommends the use of a Monte Carlo method for numerically evaluating the multi-dimensional integrals required for emulation-based sensitivity analysis instead of a quadrature-based method. This is due to the lower computational resources required as the dimensionality of the input space increases, the natural extension of the calculation of the Sobol' indices to multiple inputs instead of just one, and the ability to take account of the numerical error arising from the numerical integration estimates.

Some sensitivity analysis techniques have already been implemented in `R`. In particular, the package `‘sensitivity’` (Pujol et al., 2017) includes functions to estimate the Sobol' indices $\widehat{S}_\kappa$ using a range of methods - including those based on GP emulation - in a highly efficient manner based on the work of Le Gratiet et al. (2014). The package `‘tgp’` (Gramacy and Taddy, 2010) also contains functions for both Sobol' indices and model decomposition using GP emulation and Monte Carlo methods. However, we decided not to utilise this existing work but to write our own code instead. The main reason for this is that the extension to the case of multiple linked models can be more effectively done using bespoke code. This will be discussed further in Chapter 6.

Following Le Gratiet et al. (2014), we use a numerical method based upon multi-dimensional Monte Carlo integration (see for example Press and Farrar 1990). The integration method chosen requires only a sample from the distribution we wish to integrate with respect to (for double and triple integrals, two and three samples are required respectively).

An example of this approach to Monte Carlo integration can be seen in our method of calculation for the second term of equation (5.13),

$$\mathbf{E} = \int_{\chi_\kappa} \int_{\chi_{-\kappa}} \int_{\chi_{-\kappa}} \mu^*(\mathbf{x})\mu^*(\mathbf{x}^*)\mathrm{d}G_{-\kappa|\kappa}(\mathbf{x}_{-\kappa}|\mathbf{x}_\kappa)\mathrm{d}G_{-\kappa|\kappa}(\mathbf{x}'_{-\kappa}|\mathbf{x}'_\kappa)\mathrm{d}G_\kappa(\mathbf{x}_\kappa)\,, \tag{5.15}$$

for which we apply Algorithm 3.

Algorithm 3: Algorithm to calculate an approximate value of (5.15) using Monte Carlo integration

```
Input: Monte Carlo sample size, $M$; marginal distribution $G_\kappa$ for inputs $\mathbf{x}_\kappa$;
       conditional distribution $G_{-\kappa|\kappa}$ for inputs $\mathbf{x}_{-\kappa}$ given $\mathbf{x}_\kappa$
Output: Estimate $\hat{E}$ for the value of $\mathbf{E}$ in (5.15)
begin
    $\mathbf{R} \leftarrow$ vector of length $M$ ;
    $\mathbf{x}_\kappa \leftarrow$ sample of size $M$ from $G_\kappa$ ;
    $\mathbf{x}_{-\kappa} \leftarrow$ sample of size $M$ from $G_{-\kappa|\kappa}$ given $\mathbf{x}_\kappa$ ;
    $\mathbf{x}'_{-\kappa} \leftarrow$ sample of size $M$ from $G_{-\kappa|\kappa}$ given $\mathbf{x}_\kappa$ ; // $\mathbf{x}_{-\kappa}, \mathbf{x}'_{-\kappa}$ independent
     samples
    for $i \leftarrow 1$ to $M$ do
        $\mathbf{x}^{(i)} \leftarrow (\mathbf{x}_\kappa^{(i)}, \mathbf{x}_{-\kappa}^{(i)})^T$ ;
        $\mathbf{x}^{*(i)} \leftarrow (\mathbf{x}_\kappa^{(i)}, \mathbf{x}'^{(i)}_{-\kappa})^T$ ;
        $\mathbf{R}^{(i)} \leftarrow \mu^*(\mathbf{x}^{(i)})\mu^*(\mathbf{x}^{*(i)})$ ;          // $\mu^*()$ defined in (??)
    end
    $\hat{E} \leftarrow$ mean $(\mathbf{R})$ ;
end
```

In practice, we assume that the distributions on the inputs are independent. This allows Algorithm 3 to be simplified by drawing two samples from the full distribution G of all the inputs $\mathbf{x}$ before any sensitivity analysis is performed, and partitioning these into samples for $\mathbf{x}_\kappa$, $\mathbf{x}_{-\kappa}$ and $\mathbf{x}'_{-\kappa}$ as required.

To demonstrate our methods in action, we recreate an example given in Oakley and O'Hagan (2004). This example has 15 inputs split into three categories: the inputs $x_1, ..., x_5$ play very little role in explaining the variance in the model; the inputs $x_6, ..., x_{10}$ make a small contribution; and the inputs $x_{11}, ..., x_{15}$ explain the majority of the variance. Following the original paper, we used 250 design points to fit the Gaussian process. The expected output $E^*[E(Y|x_i)]$ for each x_i on the range $(-2, 2)$ is shown in Figure 5.1.

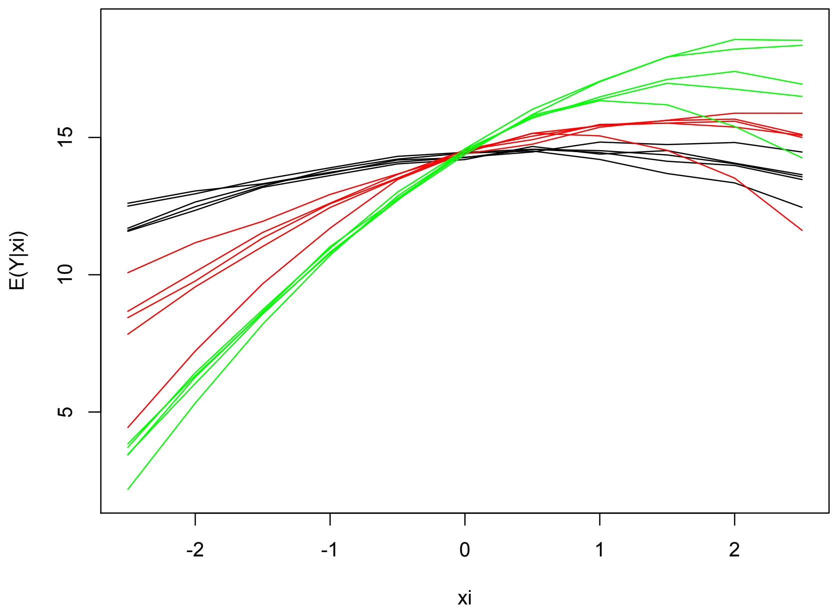

Figure 5.1: Posterior expectation of Y given x_i for $i = 1, ..., 5$ (black lines), $i = 6, ..., 10$ (red lines), $i = 11, ..., 15$ (green lines) in the test example.

This plot is extremely similar to that obtained in the original paper. The three sets of inputs can be clearly distinguished. We also calculated estimates of the Sobol' indices S_i for each x_i; in every case, our result was within 1% of that obtained in the original paper. The paper does not include content on sensitivity indices or posterior means for interaction effects, or on uncertainty in the estimates of the posterior means.

Via a small alteration to Algorithm 3, it is also possible to encompass the case where it is not possible to sample directory from the marginal and conditional distributions G_κ and $G_{-\kappa}$ and must instead use importance sampling. If, instead of samples from the distributions G_κ and $G_{-\kappa}$, we have samples from importance distributions S_κ and $S_{-\kappa}$, only one line of Algorithm 3 would change: where $\mu^*(\mathbf{x}^{(i)})\mu^*(\mathbf{x}^{*(i)})$ is calculated and stored in $\mathbf{R}$, this would be replaced with

$$\left[\frac{G_\kappa(\mathbf{x}_\kappa^{(i)})G_{-\kappa}(\mathbf{x}_{-\kappa}^{(i)})G_{-\kappa}(\mathbf{x}'^{(i)}_{-\kappa})}{S_\kappa(\mathbf{x}_\kappa^{(i)})S_{-\kappa}(\mathbf{x}_{-\kappa}^{(i)})S_{-\kappa}(\mathbf{x}'^{(i)}_{-\kappa})}\right]\mu^*(\mathbf{x}^{(i)})\mu^*(\mathbf{x}^{*(i)})\,.$$

5.5 Example: CBR modelling

We present here a relevant example of sensitivity analysis for a single model. The model considered is an illustrative example model provided by Dstl which has been simplified for release to academia. The model predicts the casualty rate in the case of a hazardous chemical, biological or radiological (CBR) release. It takes ten inputs including four categorical variables; as these lie outside of the framework introduced in previous sections, we will take these input to be fixed throughout. The six remaining inputs which we investigate are: time of release after vaccination, mass of release, wind speed, wind direction, number of people in the area, and the radius of the region in which the people are contained. The units in which these inputs are measured and the assumed lower and upper bounds on these inputs are given in Table 5.1. These ranges are identical to those used in earlier work on this model in a Masters thesis by Plumb (2008); the choice of the lower and upper bounds is explained in this work with reference to real examples.

Input	**Units**	**Lower bound**	**Upper bound**
Release time	seconds	0	72000
Release mass	kilograms	0	10000
Wind speed	metres per second	0	10
Wind direction	degrees	0	360
People	-	0	1000
Radius	metres	0	2000

Table 5.1: Assumed ranges and units for the inputs to the CBR model, chosen for consistency with previous research

Additionally, the model requires that release time, wind direction and number of people are integers. The other three inputs of interest are defined as floating point numbers. Wind direction has the additional property of existing on a circle: its lower and upper bounds are identical in the real world as they correspond to precisely the same direction. Ideally, we would take account of this in our modelling, but this was not investigated here.

The output of the model is the predicted casualties as a percentage of the number of people, rounded to an integer value. Since the output is a percentage, it exists on a bounded scale. This would pose problems when using a Gaussian process to model the output, because the assumption that the residuals from the regression are equally likely to be positive or negative would be violated close to the extremes of the scale. To overcome this, we rescale the output to a $[0, 1]$ scale instead of $[0, 100]$ and apply a logit transform, which maps the $[0, 1]$ interval to the entire real line; we then fit a Gaussian process emulator to the logit of the model output, and rescale our emulator estimates by applying the inverse transform where necessary.

The experimental designs used for each of the settings described below are maximin Latin hypercube designs with 50 points, and are obtained using the `R` package `'SLHD'` (Ba, 2015). We fit a Gaussian process emulator with a constant mean instead of a more complex regression term. When fitting the GP emulator to the logit of the model output, we first scale the inputs to the $[0, 1]$ interval. This should have no effect on the emulator we arrive at, but avoids the computational difficulties which can arise should the correlation parameter for a variable on a large scale (the release mass input, for instance) be small; it also ensures that the correlation parameters of the GP are on the same in each dimension, which makes them easier to interpret. A nugget of $\delta = 10^{-7}$ is used in the emulator to provide computational stability.

Since our inputs lie in a bounded region, the distribution G on these inputs must respect these constraints. In the case of a distribution for which some or all of the inputs follow a normal or gamma distribution, for instance, we must use truncated versions of the standard distribution. The `R` package '`truncdist`' (Novometsky and Nadarajah, 2016) allows us to do this without difficulty.

We begin by considering the CBR model with all six inputs of interest as potential sources of uncertainty, and first set the inputs to be identically distributed. The distribution we choose is a truncated normal distribution with mean 0.5 and variance 0.2. The first-order Sobol' indices for this model are given to four decimal places in Table 5.2.

Input	$\widehat{\mathbf{S}}_\mathbf{i}$
x_1	0.2051
x_2	0.0075
x_3	0.0343
x_4	0.0008
x_5	0.0010
x_6	0.7053

Table 5.2: Estimates of $\widehat{S}_i$ for each input in the full CBR model

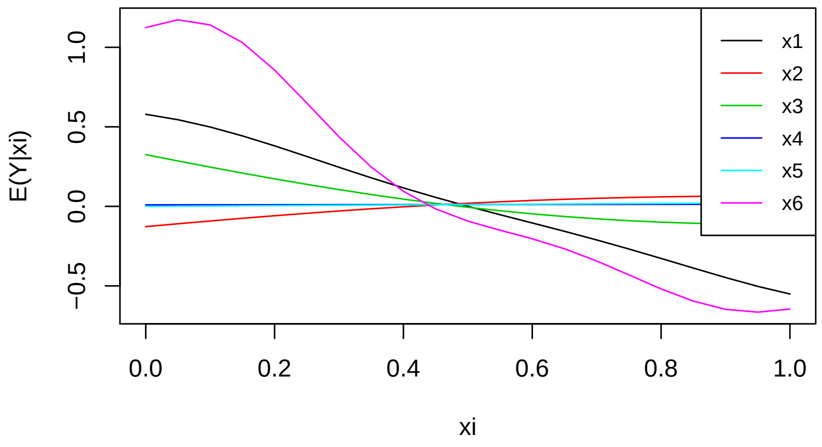

Figure 5.2: Posterior expectation of Y given each x_i in the full CBR model

The sixth input, radius of the region, appears to explain by far the largest proportion of the variability. There is also a significant contribution from x_1, the release time. To see the effects of the inputs more clearly, we can look at the plot of the posterior expectation of main effects (Figure 5.2). It is again clear that x_1 and x_6 are the most significant variables; the average effect of increasing each of these is to reduce the expected value of the proportion of casualties, although the behaviour at the extremes of radius is less clear. We can also now more clearly distinguish the effects of the other inputs: the curve for x_3, wind speed, displays a generally negative trend as the value

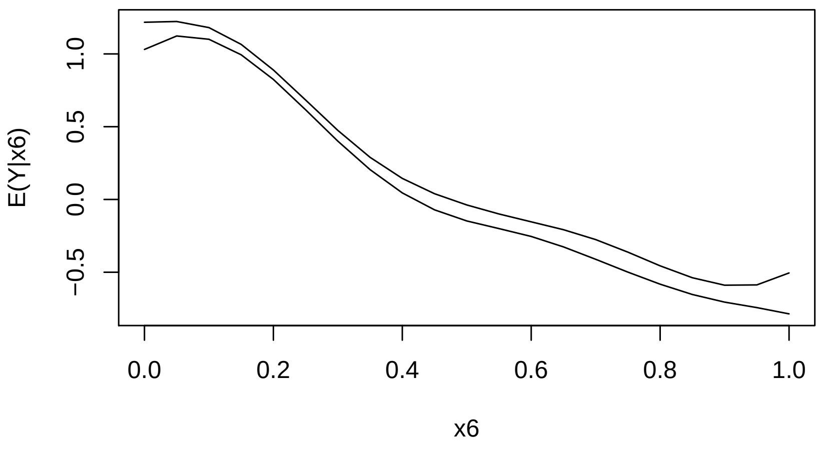

Figure 5.3: ± 2 s.d. bounds on the posterior expectation of Y given x_6 in the full CBR model

of the input is increased, while x_2, release mass, has the opposite effect. Fixing inputs x_4 or x_5 appears to tell us little about the average value of the output beyond what is already known from the emulator mean, which is consistent with the extremely low Sobol' indices obtained for these two inputs.

It is also useful to consider the uncertainty in our estimates for the expected output given a fixed input. In this case, there is relatively little variance in the estimates: in Figure 5.3, we plot the upper and lower bounds of a region within two standard deviations of the expected value of the output given x_6. While there is a little uncertainty, in particular at the extremes of the region, the shape of the function is clear.

The sum of the first-order Sobol' indices is around 0.955, meaning that around 4.5% of the output variance is not explained by main effects but by higher-order terms. We can investigate this further by looking at the sensitivity indices for the two-factor interactions directly; these are given in Table 5.3.

$\widehat{\mathbf{S}}_{\mathbf{i,j}}$	x_2	x_3	x_4	x_5	x_6
x_1	0	0.0063	0	0	0.0209
x_2	-	0	0	0	0.0060
x_3	-	-	0	0	0.0063
x_4	-	-	-	0	0
x_5	-	-	-	-	0

Table 5.3: Estimates of $\widehat{S}_{i,j}$ for each pair of inputs in the full CBR model

The sum of these effects is slightly under 0.04, so the main effects and two-factor interactions together explain nearly all of the output variation. Of the interaction effects, only the interaction between x_1 and x_6 appears to be of any real importance. This can be investigated by a two-dimensional level plot, which can be seen in Figure

5.4. This plot is not easy to interpret, but the behaviour is more complex than a simple combination of their main effects.

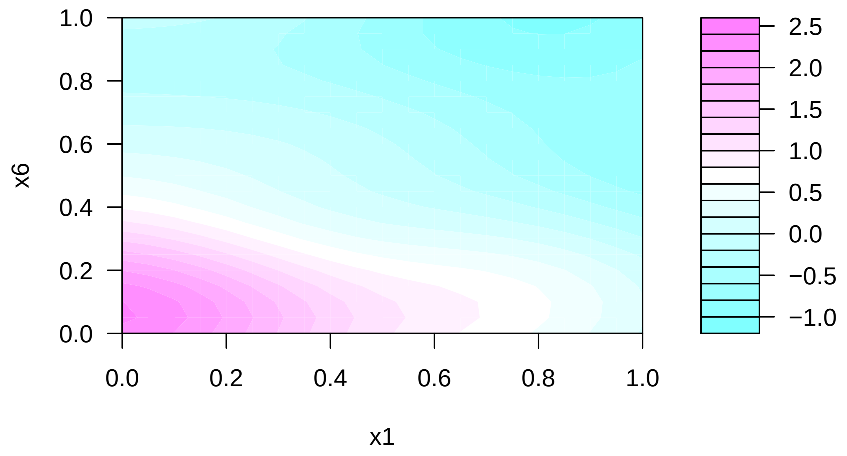

Figure 5.4: Contour plot of the posterior expectation of Y as the inputs x_1 and x_6 are varied in the full CBR model

The choice of normal distributions for the six inputs is perhaps unrealistic. There is no reason, for instance, to think that release time, release mass or radius should be less likely to take extreme values than they should to be in the middle of their respective ranges; uniform distributions may therefore be more appropriate. Wind speed is likely to follow a skewed distribution, with low values making up more of the mass of the distribution than high ones.

Following this reasoning, we rerun the above example with a different set of input distributions. The existing truncated normal distributions are maintained on inputs x_4 and x_5. Inputs x_1, x_2 and x_6 are given uniform distributions. The distribution for input x_3, wind speed, is a truncated gamma distribution with shape parameter 0.4 and scale parameter 2. This distribution is strongly positively skewed; the mean and median of a truncated gamma distribution with these parameters on the $[0, 1]$ interval are around 0.25 and 0.13 respectively.

Table 5.4 gives the Sobol' index for the main effect of each input, and Table 5.5 the Sobol' index for each two-factor interaction. Little appears to have changed as a result of this new approach, although there are some differences. The proportion of the variance explained by x_6 is somewhat reduced, and that explained by x_1 and the x_1-x_6 interaction has increased. Perhaps surprisingly, the Sobol' index for x_3, which now has a truncated gamma distribution, is also lower than in the previous example. The main effect plot (Figure 5.5) shows very little change in the curve for x_6, while the curves for the other inputs are similar in shape to those in Figure 5.2 but with their values increased throughout. This is explained by the uniform distribution on x_6, which gives more weight to points at the lower extreme of the distribution, where the output is generally high. The interaction plot for x_1 and x_6 (Figure 5.6), is similar to that seen previously (Figure 5.4).

Input	$\mathbf{\widehat{S}_i}$
x_1	0.2396
x_2	0.0077
x_3	0.0074
x_4	0.0009
x_5	0.0009
x_6	0.6672

Table 5.4: Estimates of $\widehat{S}_i$ for each input in the full CBR model with new input distributions

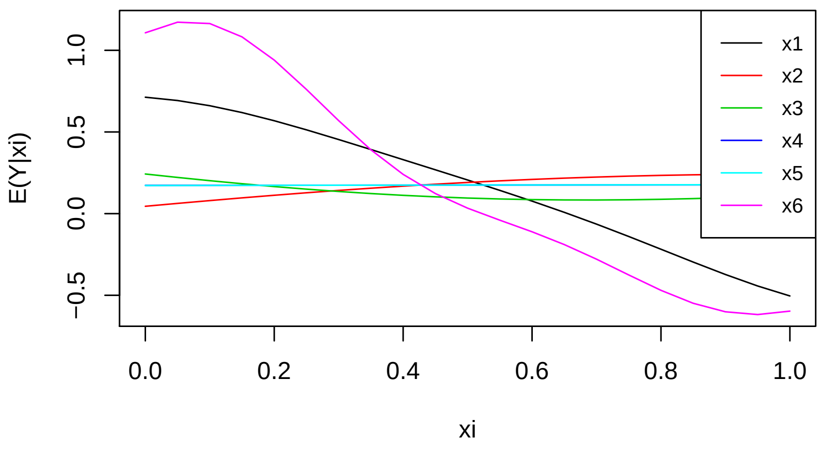

Figure 5.5: Posterior expectation of Y given each x_i in the full CBR model with new input distributions

$\mathbf{\widehat{S}_{i,j}}$	x_2	x_3	x_4	x_5	x_6
x_1	0.0005	0.0067	0	0	0.0440
x_2	-	0	0	0	0.0041
x_3	-	-	0	0	0.0073
x_4	-	-	-	0	0
x_5	-	-	-	-	0

Table 5.5: Estimates of $\widehat{S}_{i,j}$ for each pair of inputs in the full CBR model with new input distributions

Under both sets of input distributions, the relationship between the significant inputs and the output in this example is mostly linear in nature, with little role played by interactions between the inputs. Indeed, a multiple linear regression model with the

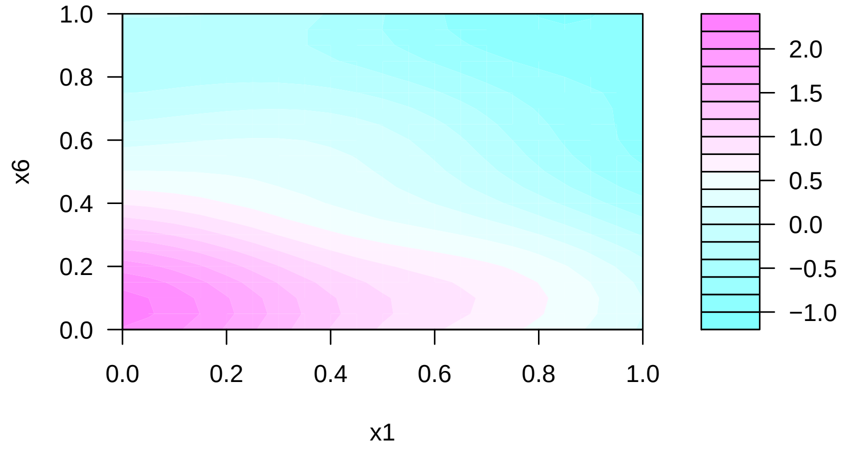

Figure 5.6: Contour plot of the posterior expectation of Y as the inputs x_1 and x_6 are varied in the full CBR model - new input distributions

six input variables as predictors is sufficient to explain over 90% of the variance in the output. There is thus little need for the emulator-based sensitivity analysis we apply here, since a much simpler and faster method is adequate.

Previous research into this model by Plumb (2008) focused on the case where the radius input is fixed, which produces more complex structures than the ones seen so far. Following this approach, we fix the radius input to the middle of its range - 1000 metres - and rerun the model with a 50-point maximin Latin hypercube design on the remaining five dimensions. As we would expect, the posterior variance in the output falls dramatically when this is done (to around 30% of its previous value), but there remains enough variability that sensitivity analysis is both meaningful and useful. There is still a significant linear relationship between the inputs and the output - linear regression explains slightly under 80% of the total variance - but there is also evidence of more complex patterns.

Input	$\widehat{\mathbf{S}}_\mathbf{i}$
x_1	0.7939
x_2	0.0608
x_3	0.0918
x_4	0.0010
x_5	0.0053

Table 5.6: Estimates of $\widehat{S}_i$ for each input in the CBR model with fixed radius

With the variable responsible for the majority of the variance in the output no longer playing a part, new patterns can be observed in the behaviour of the model with respect to the other inputs. Again beginning with truncated normal (0.5, 0.2) distributions on each of the five remaining inputs, the Sobol' indices for the main effects (Table 5.6)

$\widehat{\mathbf{S}}_{\mathbf{i,j}}$	x_2	x_3	x_4	x_5
x_1	0.0068	0.0110	0	0.0023
x_2	-	0.0051	0	0.0008
x_3	-	-	0	0.0063
x_4	-	-	-	0

Table 5.7: Estimates of $\widehat{S}_{i,j}$ for each pair of inputs in the CBR model with fixed radius

suggest that a large majority of the output variance is explained by input x_1, with smaller contributions from x_2 and x_3 and the effects of x_4 and x_5 being negligible. From Table 5.7, the only interaction term to account for more than 1% of the output variance is the interaction between inputs x_1 and x_3, the two variables with the largest main effects.

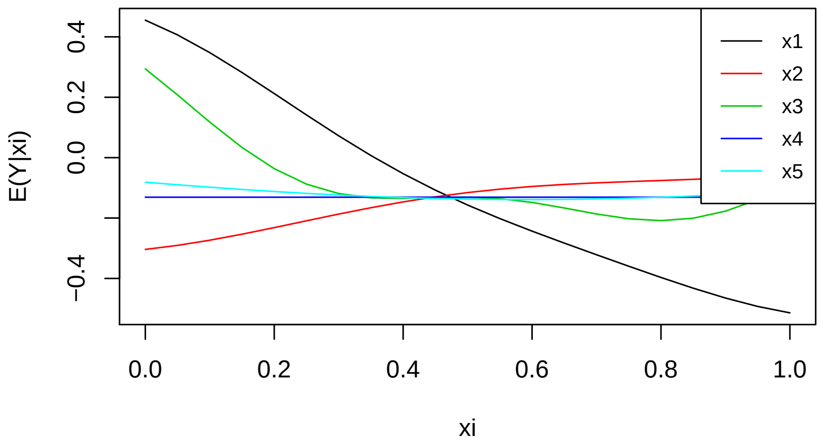

Figure 5.7: Posterior expectation of Y given each x_i in the CBR model with fixed radius

The main effects plot (Figure 5.7) displays a largely linear negative trend as x_1 is increased, with a somewhat non-linear positive trend for x_2 and a more complex effect as x_3 is varied in isolation. Looking at the uncertainty in these estimates for the two most significant variables, we observe somewhat disparate results: there is little uncertainty in the effect of fixing x_1 (Figure 5.8), but substantially wider uncertainty bounds on x_3 (Figure 5.9). This is likely to be a result of the differing amounts of uncertainty remaining in the output when the two inputs are fixed: around 21% of the output uncertainty is unexplained once x_1 is known, but around 91% is unexplained when x_3 is fixed. The uncertainty in the effect of x_3 is enough to make the true shape of the curve in Figure 5.7 unclear: the turning point in the centre (and to a lesser extent the increase at the upper extreme of the input range) may not in fact accurately reflect the effect of the input.

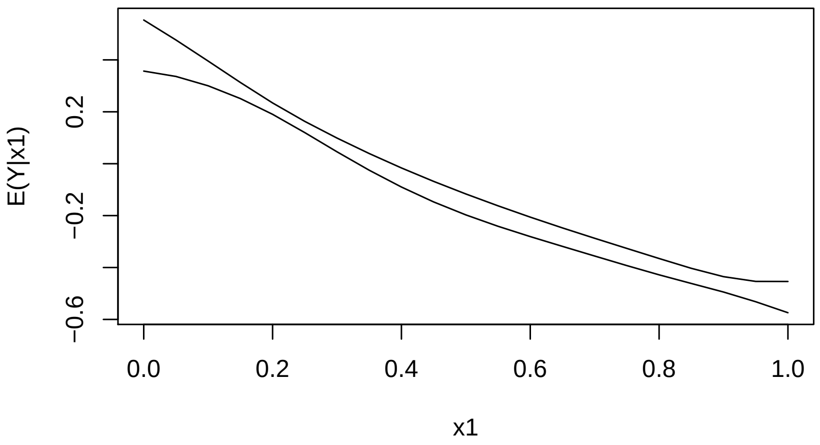

Figure 5.8: ±2 s.d. bounds on the posterior expectation of Y given x_1 in the CBR model with fixed radius

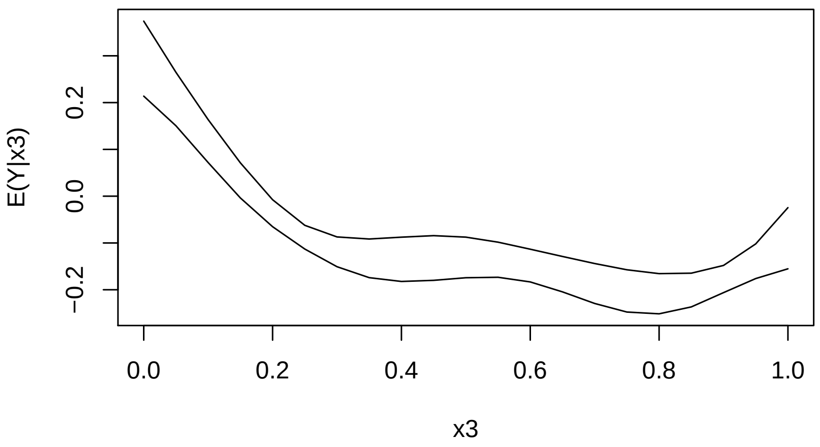

Figure 5.9: ±2 s.d. bounds on the posterior expectation of Y given x_3 in the CBR model with fixed radius

The effect of the varying x_1 and x_3 in tandem can be seen in the contour plot in Figure 5.10. The behaviour of the output here is much more complex than in previous examples, reflecting both the strong (and, in the case of x_3, non-linear) main effects of both inputs and the significance of the interaction between them. There can be little uncertainty associated with this plot, since almost 90% of the output variance is explained by the two inputs.

In this more nuanced example, the choice of input distributions can have a significant effect on the apportioning of the output uncertainty. To demonstrate this, consider again the second set of distributions used in the previous case: uniform distributions on x_1 and x_2, a truncated gamma distribution on x_3 and truncated normal distributions on x_4 and x_5. The sensitivity indices associated with this choice of distributions for the case where radius is fixed can be seen in Tables 5.8 (for main effects) and 5.9 (interactions).

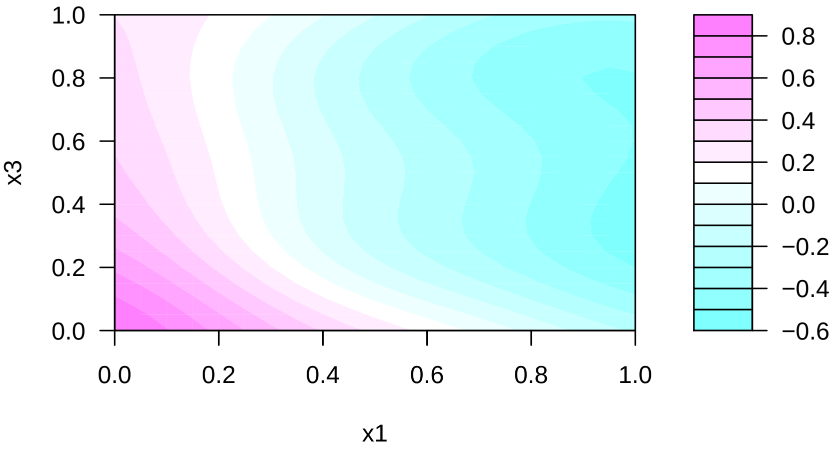

Figure 5.10: Contour plot of the posterior expectation of Y as the inputs x_1 and x_3 are varied in the CBR model with fixed radius

These are very different to those seen with truncated normal distributions on all of the inputs: the Sobol' index for the main effects of inputs x_1 and x_2 are reduced, while that associated with the main effect of x_3 increases dramatically. There are also small but noticeable increases in the indices for the x_1-x_2 and x_2-x_3 interactions.

Input	$\widehat{\mathbf{S}}_\mathbf{i}$
x_1	0.6572
x_2	0.0454
x_3	0.1985
x_4	0.0000
x_5	0.0000

Table 5.8: Estimates of $\widehat{S}_i$ for each input in the CBR model with fixed radius - new input distributions

$\widehat{\mathbf{S}}_\mathbf{i,j}$	x_2	x_3	x_4	x_5
x_1	0.0366	0.0113	0	0.0037
x_2	-	0.0191	0	0.0026
x_3	-	-	0	0.0030
x_4	-	-	-	0

Table 5.9: Estimates of $\widehat{S}_{i,j}$ for each pair of inputs in the CBR model with fixed radius - new input distributions

Considering the substantial nature of these changes, the main effects plot (Figure 5.11) is surprisingly similar to the previous plot in Figure 5.7. While the scale on which the curves move is substantially different for all inputs bar x_3, their shapes do not differ much from the previous case, with the exception of that for x_2. The uncertainty bounds on the curves for x_1 and x_3 are so similar to those in Figures 5.8 and 5.9 respectively that there is little need to include them here.

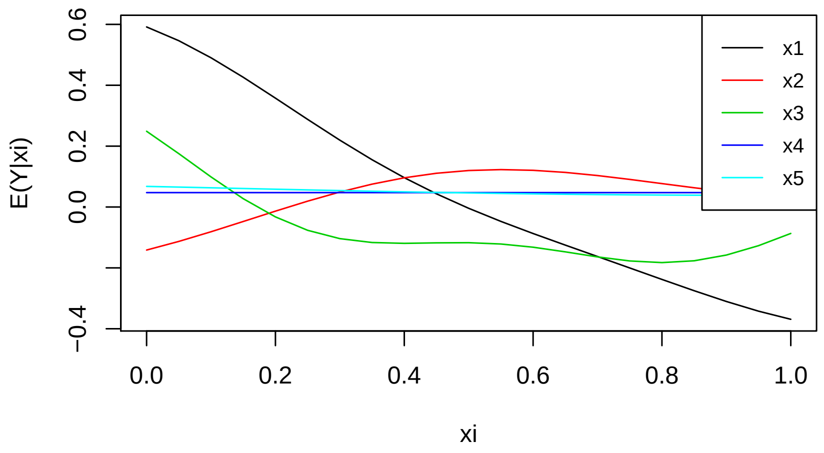

Figure 5.11: Posterior expectation of Y given each x_i in the CBR model with fixed radius - new input distributions

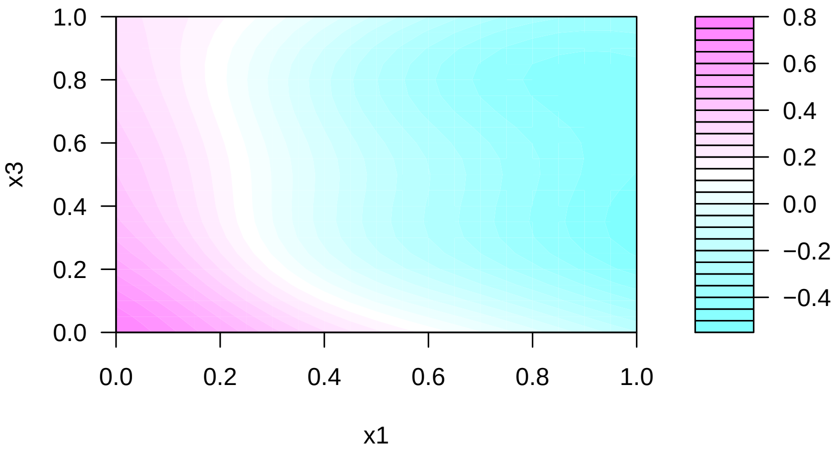

Figure 5.12: Contour plot of the posterior expectation of Y as the inputs x_1 and x_3 are varied in the CBR model with fixed radius - new input distributions.

The interaction between x_1 and x_3, investigated in the contour plot in Figure 5.12, is also much the same as that seen before (Figure 5.10). However, the two newly-significant interactions display extremely complex patterns which were not previously present. Figure 5.13 is a contour plot of the combined effect of varying x_1 and x_2. This gives us substantial insight into the behaviour of the model which could not be seen from the main effects plot or Sobol' indices. While the main effect of x_1 is roughly linear, when viewed in conjunction with x_2, new complexity emerges: the reduction in expected output as x_1 is increased is more pronounced when x_2 is close to its extreme values, and less so when x_2 is nearer to the middle of its range. The curvature in the effect of x_2 can also be seen clearly. The effect of varying x_2 and x_3 in tandem (Figure 5.12) is equally complex, although this may be more a result of these two inputs' complex main effects than the interaction between them.

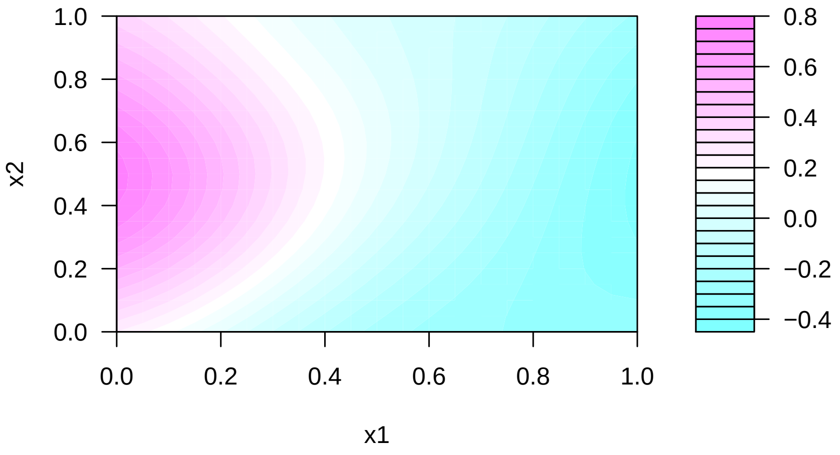

Figure 5.13: Contour plot of the posterior expectation of Y as the inputs x_1 and x_2 are varied in the CBR model with fixed radius - new input distributions.

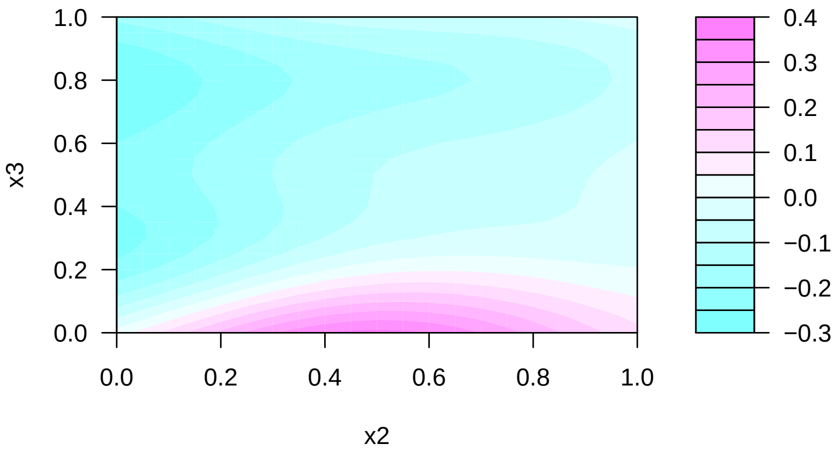

Figure 5.14: Contour plot of the posterior expectation of Y as the inputs x_2 and x_3 are varied in the CBR model with fixed radius - new input distributions.

Finally, we should note that there is some evidence that the Gaussian process emulator does not entirely capture the underlying nature of the true model. Cross-validation was performed on the two Gaussian process emulators used so far by fitting a GP emulator to 49 points in the design, predicting the output at the remaining point and comparing this to the true output. While the emulator is usually able to predict the missing point well, it struggles when the number of people is extremely low. Additional model runs reveal that the expected casualty proportion is consistently high when number of people is low, and consistently lower once it increases beyond a certain threshold (approximately 50, or 5% of the way into its range). An emulator based upon a stationary Gaussian process struggles to capture this as it assumes that the correlation in each dimension depends only on the distance between a pair of points, not on the actual values of the variable at the two, which does not appear to be the case for this input. Number of people (input x_5 in the above example) was deemed to explain a trivial proportion of the output variance in all scenarios considered here, which may not reflect its true effect on the simulator output.

5.6 Sensitivity analysis for multiple models

We now return to the two related problems in sensitivity analysis for a chain of models introduced at the beginning of this chapter, namely sensitivity analysis for the final model in a chain with respect to its own inputs, and sensitivity analysis of the full chain of models with respect to the directly controllable inputs only.

5.6.1 Sensitivity analysis for the final model in a chain with respect to the model's inputs

Sensitivity analysis on the final model is in principle the same as for a single-model case. The final model is known up to a standard Gaussian process emulator, so posterior estimates of the main effects, interactions and Sobol' indices for each of its inputs can in theory be determined using the methods described above. All of these approaches, however, require integration with respect to the distribution of the inputs to the final model. This means that there is an additional complication arising from the nature of the input y_{n-1}. While the distributions on the other inputs can be chosen using any means discussed earlier in this chapter, we have less freedom to do this for y_{n-1}, since its distribution is determined by the earlier models in the chain (and, implicitly, by the distributions of the controllable inputs to the earlier models).

This apparent stumbling block can, however, be overcome using methods already discussed in this thesis. First, we recall that the Monte Carlo integration approach introduced in Section 5.4 requires only a sample from the joint distribution of the inputs to the model; we are not required to have an analytical form for this distribution. Assuming that the inputs to the final model are independent, and that the distributions for its controllable inputs are known, sensitivity analysis can therefore be conducted if we can obtain a sample from the distribution of y_{n-1}.

In Chapter 3, we introduced two methods for prediction from a chain of multiple models. This chain can be of any length, so these methods can be applied to a chain of $n-1$ models instead of n. Thus, it is possible to predict for y_{n-1} given the inputs to the first $n-1$ models. This can be used to generate a sample from the empirical distribution for y_{n-1}. First, draw a sample from the joint distribution of $\tilde{\mathbf{x}}_1, ..., \tilde{\mathbf{x}}_{n-1}$; as in previous sections, this can be simplified by assuming the inputs are independent. For each input configuration, draw a single value for y_{n-1}. For the simulation method, this means running each input configuration once and storing the result. For the theoretical method, the mean and variance should be calculated, and a single value drawn from the implied approximate normal distribution. The result is a sample from the distribution of y_{n-1}, which can then be used for sensitivity analysis on y_n given the inputs to model n.

A simpler result holds in the case of a chain of two models. Here, to conduct sensitivity analysis for model 2, a distribution is required for the uncontrollable input y_1. Since y_1 is the output of a model which can be approximated directly by a Gaussian process emulator, a probability distribution for y_1 is available at each possible configuration of its inputs $\tilde{\mathbf{x}}_1$. A distribution for y_1 can therefore be constructed directly if the joint distribution of its inputs is known.

While the inputs to the first $n-1$ models in the chain, $\tilde{\mathbf{x}}_1, ..., \tilde{\mathbf{x}}_{n-1}$, do not appear in the final sensitivity analysis for y_n, they nonetheless play a significant role in its results. The implied distribution on y_{n-1} depends strongly on the choices of the distributions from which the samples of $\tilde{\mathbf{x}}_1, ..., \tilde{\mathbf{x}}_{n-1}$ are drawn. As seen in Section 5.5, changing the input distributions in a single model can significantly affect the sensitivity analysis results, so obtaining a reasonable empirical distribution for y_{n-1} is extremely important. It is therefore crucial that the distributions of the controllable inputs to earlier models are chosen in a way which reflects their true behaviour.

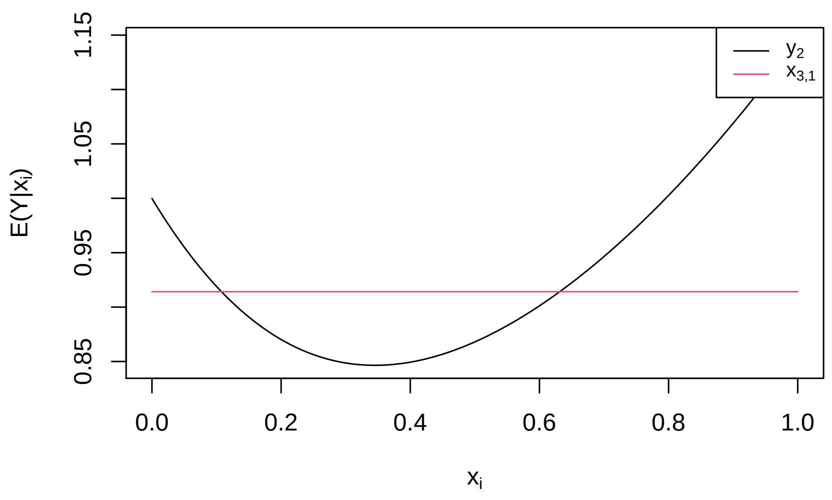

Figure 5.15: Posterior mean for y_3 given y_2 (black) and $x_{3,1}$ (red) in the simple three-model example chain

To test this technique, consider the three-model chain introduced in equation (3.17). This chain is a function of a single input $x_{1,1}$, so sensitivity analysis is not meaningful as all of its variation is explained by this single input. However, the chain could be rewritten to include several dummy inputs which do not affect the final output:

$$y_1(x_{1,1}) = x_{1,1} + sin(3\pi x_{1,1})\,, 0 \leq x_{1,1} \leq 1\,;$$

$$y_2(y_1, x_{2,1}) = y_1 - \log(1 + y1)\,;$$

$$y_3(y_2, x_{3,1}) = y_2 + \exp(-2y_2)\,. \tag{5.16}$$

Sensitivity analysis on model 3 can be conducted with respect to its inputs, y_2 and $x_{3,1}$, using the technique described above. For simplicity, we assume a uniform distribution on each of the controllable inputs to the chain. For $x_{1,1}$, the upper and lower bounds of the uniform distribution are 0 and 1 respectively. For $x_{2,1}$, the range is taken to be same as that seen in the output of the simulator runs for y_1; this avoids biasing our results by having the two variables be on vastly different scales, since we do not use scaled inputs for this chain (as will be done in Chapter 7 for the real example). Similarly, the range for $x_{3,1}$ is chosen using the simulator runs for y_2.

We fit emulators with constant regression terms and Gaussian correlation functions to the three models, and use the theoretical method for prediction from a two-model chain to obtain a sample from the distribution of y_2. 10 design points for each model are chosen from a maximin Latin hypercube design, with algorithm 2 applied to ensure the designs for y_1 and y_2 are reasonable. Standard results from Sections 5.3 and 5.4 then allow posterior estimates of the main effects, interaction and Sobol' indices to be determined.

The posterior means of the main effects of the two inputs can be seen in Figure 5.15. The results conform entirely to what would be expected for a model in which only one input has an effect. Fixing the input $x_{3,1}$ provides no information about the expected output of y_3, thus producing a flat line at the unconditional expected value of y_3, while fixing y_2 recreates exactly the shape of the true function given in equation (5.16). The Sobol' index for y_2 is $S_{y2} = 1$ to five decimal places, while the index for $x_{3,1}$ is $S_{x31} = 0$, and index for the interaction between the two inputs is zero to five decimal places. This tells us that all of the variation in y_3 is explained by y_2, which we know is true for this model.

This form of sensitivity analysis is limited, however, as it tells us little about the effects of the inputs $x_{1,1}$, $x_{1,2}$ and $x_{2,1}$ on y_3. It is useful to know that $x_{3,1}$ plays no role in explaining the variation in y_3, and it is useful to know (through the posterior expectation of the main effect) the relationship between y_2 and y_3, but this is by no means the full story. From this analysis alone, it would not be possible to determine that $x_{1,1}$ explains all of the variation in y_3, and that $x_{1,2}$ and $x_{2,1}$ play no role. The same issue applies for chains in which more than one (and potentially all) of the controllable inputs have an effect on the final output; analysis of the final model provides no information about the relative importance of the inputs to earlier models. For this reason, a full sensitivity analysis for the chain as a whole is more informative.

5.6.2 Sensitivity analysis for the final output of a chain with respect to the controllable inputs

When sensitivity analysis is required on the entire chain, the situation becomes significantly more complex. We are still interested in many of the same quantities as for the single-model case, but the results described in the above sections no longer hold. Let E^* denote an expectation with respect to the linked emulator, and $\mathbf{x} = (\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, ..., \tilde{\mathbf{x}}_n)^T$ be the set of all directly controllable inputs to the chain of models. To assess, for example, the main effects and interaction effects of the input variables, we are still interested in

$$E^*[E(Y|\mathbf{x}_\kappa)] = E^* \int_{\chi_{-\kappa}} \eta(\mathbf{x}) dG_{-\kappa|\kappa}(\mathbf{x}_{-\kappa}|\mathbf{x}_\kappa) \tag{5.17}$$

but the result given in equation (5.4) for a single GP emulator no longer holds. The additional complexity of a linked emulator framework - in which there are, for example, multiple correlation matrices coming from each individual emulator, all of which refer to different sets of inputs - means that no simple closed-form expression for $E^*[E(Y|\mathbf{x}_\kappa])$ can be found, as the outer expectation E^* is now with respect to a much more complicated function.

Clearly, the only solution to this is to estimate the integral in (5.17) using another method. The most natural way to do this is a numerical approximation by direct simulation from the linked emulator. Since the linked emulator is substantially quicker to run than the chain of models, this is more computationally feasible than running the models themselves, and is a sensible choice where no theoretical results about the linked emulator exist. For long chains with relatively large input spaces, however, this method quickly becomes infeasible, as simulation from the linked emulator is slower than that for a single GP emulator.

The theoretical results for the mean and variance of the linked emulator output introduced in Chapter 3 provide a partial alternative. Using our notation for sensitivity analysis, the posterior mean of the emulator output at an input set $\mathbf{x}$ can be written as $E^*[\eta(\mathbf{x})]$. Let $F[\eta(\mathbf{x})]$ be the distribution of $Y = \eta(\mathbf{x})$ under the linked emulator. We have

$$\begin{aligned}
E^*[E(Y|\mathbf{x}_\kappa)] &= E^* \int_{\chi_{-\kappa}} \eta(\mathbf{x}) dG_{-\kappa|\kappa}(\mathbf{x}_{-\kappa}|\mathbf{x}_\kappa) \\
&= \int_{-\infty}^{\infty} \int_{\chi_{-\kappa}} \eta(\mathbf{x}) dG_{-\kappa|\kappa}(\mathbf{x}_{-\kappa}|\mathbf{x}_\kappa) dF[\eta(\mathbf{x})] \\
&= \int_{\chi_{-\kappa}} \int_{-\infty}^{\infty} \eta(\mathbf{x}) dF[\eta(\mathbf{x})] dG_{-\kappa|\kappa}(\mathbf{x}_{-\kappa}|\mathbf{x}_\kappa) \\
&= \int_{\chi_{-\kappa}} E^*\eta(\mathbf{x}) dG_{-\kappa|\kappa}(\mathbf{x}_{-\kappa}|\mathbf{x}_\kappa) \\
&= E[E^*(Y|\mathbf{x}_\kappa)] \,. \qquad (5.18)
\end{aligned}$$

where the third equality holds because, as in the single-emulator case (see for example Oakley and O'Hagan, 2004), the distribution of $\hat{Y}(\mathbf{x})$ is independent of that of $\mathbf{x}_{-\kappa}|\mathbf{x}_\kappa$ so the order of integration can be changed by a special case of Fubini's theorem. We can thus use the theoretical value for $E^*[\eta(\mathbf{x})]$ to calculate the posterior expectations of the main effects and interactions of the inputs to a linked emulator. A numerical method is still required for the integral over $\chi_{-\kappa}$, but this can be done using essentially the same method described in Section 5.4. Similarly, we can obtain the expectation of Y with no fixed inputs by

$$E^*[E(Y)] = \int_\chi E^*[\eta(\mathbf{x})] dG(\mathbf{x}) \,.$$

Consider a two-model chain consisting of the models

$$y_1(x_{1,1}) = x_{1,1} + sin(3\pi x_{1,1}) \,, 0 \le x_{1,1} \le 1 \,;$$

$$y_2(y_1, x_{2,1}) = y_1 - \log(1 + y1) \,;$$

This is a subset of the chain defined in equation (5.16) in which the third model has been removed. It depends only on its first input $x_{1,1}$; the input $x_{2,1}$ does not affect the output y_2. Using the method described above, we can generate a plot of the posterior mean of y_2 given $x_{1,1}$ and $x_{2,1}$ with respect to the linked emulator. This is shown in Figure 5.16.

The plot demonstrates that fixing $x_{2,1}$ provides no information about y_2, and shows the relationship between $x_{1,1}$ and y_2 exactly. For a model with multiple important inputs, however, the plot may be less easy to interpret, and could be misleading as the variance is not taken into account.

The situation for the variance of the main effect or interaction,
$Var^*[E(Y|\mathbf{x}_\kappa)]$, is more complicated. Since

$$Var^*[E(Y|\mathbf{x}_\kappa)] = E^*\{[E(Y|\mathbf{x}_\kappa)]^2\} - \{E^*[E(Y|\mathbf{x}_\kappa)]\}^2 \,,$$

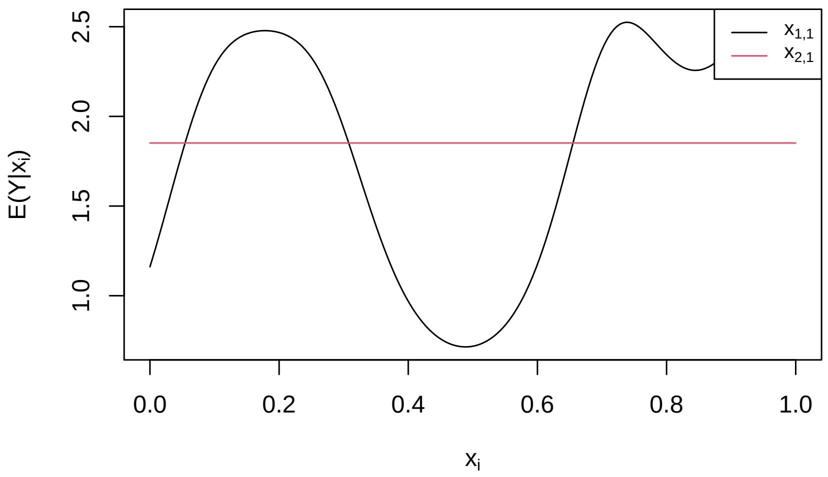

Figure 5.16: Posterior mean with respect to the linked emulator for y_2 given $x_{1,1}$ (black) and $x_{2,1}$ (red) in a subset of the three-model example chain containing only the first two models

and the second term is the square of what was already calculated in (5.18), we require an expression for $E^*\{[E(Y|\mathbf{x}_\kappa)]^2\}$. This is equivalent to

$$E^*\{[E(Y|\mathbf{x}_\kappa)]^2\} = E^*\left\{\left[\int_{\mathcal{X}_{-\kappa}} \eta(\mathbf{x})dG_{-\kappa|\kappa}(\mathbf{x}_{-\kappa}|\mathbf{x}_\kappa)\right]^2\right\},$$

but here the outer expectation with respect to the linked emulator cannot be taken inside the integral due to the presence of the squared term. An alternative would be to note that

$$[E(Y|\mathbf{x}_\kappa)]^2 = Var(Y|\mathbf{x}_\kappa) - E[(Y|\mathbf{x}_\kappa)]^2$$

and attempt to obtain expressions for $E^*[Var(Y|\mathbf{x}_\kappa)]$ and $E^*\{E[(Y|\mathbf{x}_\kappa)]^2\}$. The same problem arises again, however, and no theoretical result for these expressions is possible given only $E^*[\eta(\mathbf{x})]$ and $Var^*[\eta(\mathbf{x})]$.

In Section 5.3, the variance of the main effect or interaction was found using the general result in (5.10), attributable to Oakley and O'Hagan (2004). This result requires knowing the covariance between the output of two independent realisations of the GP emulator, so a similar approach could in theory be used for a linked emulator. A derivation of the same covariance for the case of a linked emulator has to date not been possible; this is discussed further in Chapter 9.

To obtain an estimate of the posterior expectation of the sensitivity indices for a set of inputs $\mathbf{x}_\kappa$, we require

$$\hat{S}_\kappa = \frac{E^*\{Var[E(Y|\mathbf{x}_\kappa)]\}}{E^*[Var(Y)]}.$$

As was the case for a single GP emulator, this is not an exact expression for S_κ since the ratio of an expectation does not equal the expectation of a ratio. To calculate the denominator, note that

$$\begin{aligned}
E^*[Var(Y)] &= E^*[E(Y^2)] - E^*\{[E(Y)]^2\} \\
&= E[E^*(Y^2)] - E^*\{[E(Y)]^2\} \\
&= E[Var^*(Y)] + E\{[E^*(Y)]^2\} - E^*\{[E(Y)]^2\} \\
&= E[Var^*(Y)] + E\{[E^*(Y)]^2\} - Var^*[E(Y)] - \{E^*[E(Y)]\}^2 \\
&= E[Var^*(Y)] + Var[E^*(Y)] - Var^*[E(Y)] \qquad (5.19)
\end{aligned}$$

Again, however, this is dependent on $Var^*[E(Y)]$, for which no result is currently available. Similar issues arise when attempting to derive an expression for the numerator, and no result for this currently exists. The case of sensitivity analysis for a chain of emulators which do not satisfy the constraints required for the theoretical means and variances to be calculated also remains open unless direct simulation can be used.

5.7 Conclusions

This chapter reviews existing approaches to sensitivity analysis for a single computational model, and presents approaches to extend these to a chain of models. We begin with a broad view of sensitivity analysis in its many forms, before focusing on the specifics of model decomposition and variance-based sensitivity analysis, and how they can be used within the framework of Gaussian process emulation. These techniques were then applied to a model for casualties of a CBR release, demonstrating the valuable insights which can be provided by sensitivity analysis in understanding the behaviour of a model with respect to its inputs. This example is directly relevant to our eventual research goals: Chapter 7 also deals with casualties from a CBR release, but using a chain of models instead of a "one-hit" setup.

Our attempts to create methods for sensitivity analysis for a chain of models were only partially successful. Nevertheless, a concrete method for sensitivity analysis for the final model in the chain is a valuable tool. It is noteworthy that this method is only possible when combined with Monte Carlo integration for a single emulator; the use of an alternative numerical method such as quadrature would complicate matters significantly, since only an empirical distribution is available for the input y_{n-1}. As discussed in Section 5.4, however, there are already several practical advantages to the use of a Monte Carlo approximation to the unknown integrals even for the simpler single-model case.

With regards to sensitivity analysis for the chain as a whole, there is still much work to be done. Posterior estimates of the main effects and interactions of the directly controllable inputs are undoubtedly useful, but limited in scope. No quantification of the uncertainty in these estimates is possible, and reliable estimates of the Sobol' indices cannot be derived from them; the naive approach of simply estimating the Sobol' index of an input i by

$$E^*(V_i) \approx Var\{E^*[E(Y|x_i)]\}$$

is not recommended since it takes no account of the variance in the estimate of the main effect. A full derivation would appear to depend on a theoretical result for the posterior covariance between two outputs from the final model under the linked emulator, which has not so far been possible to determine, although the beginnings of a derivation are given in Chapter 9.

Additionally, the posterior estimate of the main effects and interactions presented here is only possible when closed-form expression for the mean and variance of the final model output under the linked emulator are available, which is only the case under the fairly restrictive set of assumptions given in Theorem 3.1. In terms of sensitivity analysis for chains of emulators which do not satisfy these assumptions, there are several ideas which may be of interest, but these require substantial further theoretical and practical development. They are discussed further in Chapter 9.

Chapter 6

Software implementation

6.1 Introduction

The computer code associated with this research project, which was developed independently, is written predominantly in the `R` programming language. Its chief function is to provide methods for prediction from a chain of Gaussian process emulators. (Methods for a single GP emulator are also implemented, but these are not discussed here as their primary use is as part of the process used for a chain.) Both the Monte Carlo integration-based methods for chains of arbitrary length (see Sections 3.3 and 3.5) and the theoretical methods to obtain the mean and variance of the linked emulator (see Sections 3.4 and 3.6) are supported in our code.

The Monte Carlo method is computationally expensive to conduct, and an implementation entirely in `R` was found to be infeasibly slow even for relatively small problems. For this reason, much of the linear algebra and basic calculations required were moved to the faster `C++` language. The `R` packages '`Rcpp`' (Eddelbuettel, 2013) and '`RcppArmadillo`' (Eddelbuettel and Sanderson, 2014) are used to interface between the two languages. The code requires that the vector of regression coefficients in each of the emulators is integrated out, and that the correlation parameters are estimated using a plug-in method. The process variance may be dealt with in either manner; although these two cases lead to substantially different distributions on the emulator output at each stage, our code was developed to support both methods. The correlation function may be one of three standard types: Gaussian, Matérn with smoothness parameter $\omega = 5/2$ and Matérn with smoothness parameter $\omega = 3/2$. These were chosen due to their prior use in single-model emulation packages such as '`DiceKriging`' (Roustant et al., 2012).

The theoretical approximation to the linked emulator also makes use of `C++`, despite being relatively fast to calculate, as this allows large problems to be tackled in reasonable time. Separate functions are used for the mean and variance at a prediction point, and are both called by another function which calculates both. Due to the assumptions required for the theoretical results to hold, only a Gaussian correlation may be specified

, and the regression coefficients and process variance must be single values specified by the user.

In addition, our code also includes methods for sensitivity analysis for the final model in a chain. This combines the multiple model prediction methodology (to obtain a sample from the distribution for the input y_{n-1}) with traditional emulation-based sensitivity analysis for a single model.

The full set of functions used in our work are available on Github at https://github.com/StephenGow/Thesis-Code.

The remainder of this chapter focuses on the practical uses of the code. The vast majority of the 23 `R` and 34 `C++` functions used are intended for "behind-the-scenes" calculations, and should not need to be called directly by an end user. There are four `R` functions which should be used directly - one each for prediction from a linked emulator using simulation, prediction from a linked emulator using the theoretical method, sensitivity analysis for the final model in a chain, and posterior expectations of the final output of a chain given the directly controllable inputs. These are described in detail in Section 6.2, with the code required for their usage in an example chain presented in Section 6.3.

6.2 Details of usage

6.2.1 Prediction from a linked emulator using simulation

Function Name

`MM_pred_simu`

Description

Prediction from a linked emulator via the simulation method.

Arguments

- `sampsize`: Monte Carlo sample size.
- `predpts`: List of matrices of the prediction points for the controllable inputs to each model.
- `loc_vec`: Vector of locations of the variable y_{k-1} in the emulator for y_k.
- `designs`: List of design matrices for the individual emulators.
- `b_vecs`: List of vectors of correlation parameters for the individual emulators.
- `res_vecs`: List of vectors of results from the simulator runs for each computer experiment.

- `F_mats`: List of matrices of regression functions for the individual emulators.
- `corr_mats`: List of correlation matrices for the individual emulators.
- `cov_type`: String determining the correlation function of the emulators. Options are a Gaussian correlation ("Gaussian"), Matérn correlation with $\omega = 3/2$ ("Matern_32"), and Matérn correlation with $\omega = 5/2$ ("Matern_52").
- `scale_params`: Optional list of location and scale shift parameters for the second and later emulators in the chain.

Notes

This function assumes that the regression coefficients and process variance are integrated out, and thus samples from a t-distribution for the output of each model in the chain.

For the `predpts` input, entry k of the list should be a matrix containing the prediction locations for the inputs $\tilde{\mathbf{x}}_k$. If model k takes no inputs except y_{k-1}, entry k of the list should be set to null.

For a chain consisting of r models, the `loc_vec` input is a vector of length $r-1$ containing the locations of the variable y_{k-1} in the emulator for y_k. For example, "1" corresponds to a two-model chain in which y_1 is the first variable in the emulator for y_2.

The `scale_params` input should be used when the output y_k is on a different scale to the one used in the emulator for y_{k+1}. Default corresponds to a location shift of 0 and a scale shift of 1.

Value

A matrix of outputs from the linked emulator, where each row corresponds to a distinct prediction point.

6.2.2 Prediction from a linked emulator using the theoretical method

Function Name

`MM_pred_theory`

Description

Prediction from a linked emulator via the theoretical method.

Arguments

- `predpts`: List of matrices of the prediction points for the controllable inputs to each model.
- `designs`: List of design matrices for the individual emulators.
- `b_vecs`: List of vectors of correlation parameters for the individual emulators.
- `res_vecs`: List of vectors of results from the simulator runs for each computer experiment.
- `inv_corrmats`: List of inverted correlation matrices for the individual emulators.
- `beta0s`: List of regression coefficients for the individual emulators.
- `sig2s`: List of process variances for the individual emulators.
- `nuggets`: List of nuggets for the individual emulators.
- `scale_params`: Optional list of location and scale shift parameters for the second and later emulators in the chain.

Notes

The correlation functions of all of the individual GP emulators are required to be Gaussian. The regression components of the emulators must be constant terms instead of linear. Each y_{k-1} must be the first variable in the emulator for y_k.

For the `predpts` input, entry k of the list should be a matrix containing the prediction locations for the inputs $\tilde{\mathbf{x}}_k$. If model k takes no inputs except y_{k-1}, entry k of the list should be set to null.

The `scale_params` input should be used when the output y_k is on a different scale to the one used in the emulator for y_{k+1}. Default corresponds to a location shift of 0 and a scale shift of 1.

Value

List of two elements: the mean of the output under the linked emulator at each prediction point, and the corresponding variance at each prediction point.

6.2.3 Sensitivity analysis for the final model in a chain

Function Name

`SensFinal`

Description

Sensitivity analysis on the final model in a chain using the theoretical method for prediction.

Arguments

- `designs`: List of design matrices for the individual emulators.
- `b_vecs`: List of vectors of correlation parameters for the individual emulators.
- `res_vecs`: List of vectors of results from the simulator runs for each computer experiment.
- `inv_corrmats`: List of inverted correlation matrices for the individual emulators.
- `beta0s`: List of regression coefficients for the individual emulators.
- `sig2s`: List of process variances for the individual emulators.
- `nuggets`: List of nuggets for the individual emulators.
- `samples1`: List of matrices containing samples from the distribution for the controllable inputs to each model.
- `samples2`: List of matrices containing samples from the distribution for the controllable inputs to each model.
- `seq_length`: Number of points in each variable to evaluate the main effects and (if specified) bounds and interactions at. Default is 200.
- `bounds`: Optional numeric vector containing the variable(s) for which posterior variance bounds of ± 2 standard deviations on the expected output across the range of the variable(s) should be plotted.
- `ints`: Optional matrix containing the pairs of variables for which the two-way interaction between them should be plotted.
- `lim_low`: Optional vector of lower limits for the variables of the final model. If not supplied, defaults to 0 for all variables.
- `lim_up`: Optional vector of upper limits for the variables of the final model. If not supplied, defaults to 1 for all variables.
- `varnames`: Optional vector of variable names.

Notes

This function works by using `MM_pred_theory` to obtain a sample for the input to the final model which arises as the output of the previous model in the chain, then applying standard techniques for sensitivity analysis on a single emulator using Monte Carlo integration. It assumes independence between the variables in the final model, so that their joint distribution is a combination of their one-dimensional distributions. A Gaussian correlation function and constant regression component for the individual GP emulators in the chain is required.

By default, the function plots the posterior mean of the expected output across the range of each variable in the final model, and calculates Sobol' indices for each main effect and two-way interaction. The user may specify additional plots using the `bounds` and `ints` arguments. The `bounds` input is a vector containing the location of the relevant variable(s) in the emulator for the final model. If s interactions are required, the dimension of the matrix input `ints` should be $s \times 2$, where each row contains the locations of the relevant pair of variables in the emulator for the final model. When multiple plots are requested, they are separated by a user prompt to press the return key.

The matrices of samples contained within the list arguments `samples1` and `samples2` must be independent. If model k takes no inputs except y_{k-1}, entry k of these lists should be set to null.

The `scale_params` input should be used when the output y_k is on a different scale to the one used in the emulator for y_{k+1}. Default corresponds to a location shift of 0 and a scale shift of 1.

The `varnames` argument controls the names used for each variable in plots of the main effects and (if specified) bounds and interactions. The default is $x_1, x_2, ..., x_{q*}$, where q^* is the number of variables in the final model and consists of q_r directly controllable inputs plus y_{r-1}.

Value

List of two elements: a vector of the Sobol' indices for the individual variables, and a matrix of Sobol' indices for the two-way interactions.

6.2.4 Sensitivity analysis for the output of a chain in terms of the directly controllable inputs

Function Name

`Multimod_calcME`

Description

Posterior means of the expected output of a chain of two models as each of the directly controllable inputs to the chain is fixed to multiple values across its range.

Arguments

- `samples`: Matrix containing a sample from the joint distribution of all controllable inputs to the chain.
- `xn_1`: Design matrix for the first emulator.
- `xn_2`: Design matrix for the second emulator.
- `y1`: Vector of results from the simulator runs for the first computer experiment.
- `y2`: Vector of results from the simulator runs for the second computer experiment.
- `beta0_1`: Regression coefficient for the first emulator.
- `beta0_2`: Regression coefficient for the second emulator.
- `inv_corrmat1`: Inverted correlation matrix for the first emulator.
- `inv_corrmat2`: Inverted correlation matrix for the second emulator.
- `b1`: Vectors of correlation parameters for the first emulator.
- `b2`: Vectors of correlation parameters for the second emulator.
- `sig2_1`: Process variance for the first emulator.
- `sig2_2`: Process variance for the second emulator.
- `nugget_1`: Nugget of the first emulator.
- `nugget_2`: Nugget of the second emulator.
- `seq_length`: Number of points in each variable to evaluate the main effects and (if specified) bounds and interactions at. Default is 200.
- `lim_low`: Optional vector of lower limits for the variables of the final model. If not supplied, defaults to 0 for all variables.
- `lim_up`: Optional vector of upper limits for the variables of the final model. If not supplied, defaults to 1 for all variables.
- `varnames`: Optional vector of variable names.
- `scale_params`: Optional vector of location and scale shift parameters for the second emulator in the chain.

Notes

Independence between the controllable inputs of the chain is assumed, so that their joint distribution is a combination of their one-dimensional distributions. A Gaussian correlation function and constant regression component for the individual GP emulators in the chain is required.

In addition to its value, this function plots the posterior mean of the expected output across the range of each variable.

The `scale_params` input should be used when the output y_1 is on a different scale to the one used in the emulator for y_2. Default corresponds to a location shift of 0 and a scale shift of 1.

The `varnames` argument controls the names used for each variable in plots of the main effects and (if specified) bounds and interactions. The default is $x_1, x_2, ..., x_q$, where $q = q_1 + q_2$ is the total number of directly controllable inputs to the chain.

In the longer term, it is hoped to extend this function to support chains of more than two models, and to bring its inputs into line with the previous three functions described.

Value

Matrix of expected outputs across the range of each variable to the chain. Each row corresponds to a variable; each column corresponds to the location of the fixed value for each variable within the sequence.

6.3 Examples

We present in this section an example of how the functions described in Section 6.2 can be used for prediction and sensitivity analysis for a chain of models. The chain we shall demonstrate our `R` code on is the three-model example presented in equation (5.16), with a single important input and two dummy inputs which have no effect on the output y_3. Note that where it is necessary to break a single line of code across multiple lines of text, we shall use a tab at the beginning of the second and later lines to indicate that these should be read as part of the preceding line of code.

We begin by inputting the three models into `R`.

```
test_func1 <- function(x1){
  return(x1 + sin(3 * pi * (x1)))
}
```

```
test_func2 <- function(x1, x2){
  return(x1 - log(1 + x1))
}

test_func3 <- function(x1, x2){
  return(x1 + exp(-2*x1))
}
```

We now need to run computer experiments for each of these models, and extract the relevant values for use in the linked emulator. Before doing this, however, we define a function to set up the matrix of correlations between the design points of a computer experiment, as this will be useful later.

```
corr_mat_setup <- function(corr_func, design, b, nugget){
  design <- as.matrix(design)
  npts <- nrow(design)
  corr_mat <- matrix(nrow=npts, ncol=npts)
  for(i in 1:npts){
    for(j in 1:npts){
      corr_mat[i, j] <- corr_func(b, design[i,] - design[j,])
    }
  }
  corr_mat <- corr_mat + diag(nugget, npts)
  return(corr_mat)
}
```

To allow the theoretical linked emulator to be used, a Gaussian correlation function and a constant regression term are used. The experimental designs are chosen as described in Section 5.6. A nugget $\delta = 10^{-7}$ is used in each emulator. The `mlegp` package is used to fit the emulators. For the first model, the code required to run the computer experiment, set up the matrix of regression functions and nugget, build the GP emulator, extract the correlation parameters and populate the matrix of correlations is as follows:

```
library(mlegp)

npts_1 <- 10
xn_1 <- seq(0, 1, len=npts_1)
y1 <- vector(length=npts_1)
for (i in 1:npts_1){
  y1[i] <- test_func1(xn_1[i])
}

F_mat1 <- matrix(rep(1, npts_1))
nugget_1 <- 1e-7
GP_1 <- mlegp(xn_1, y1)
b1 <- GP_1$beta
corr_mat1 <- corr_mat_setup(corrfunc_Gauss_Cpp, xn_1,  b1,
        nugget_1)
```

The function `corrfunc_Gauss_Cpp` is a `C++` function to calculate the correlation between two points under a Gaussian correlation model, which is not repeated here for reasons of space. With some slight alterations to account for the second and third model being (nominally) functions of two inputs, and an adaptation in the experimental design procedure as described in Algorithm 2, the code for the remaining two models is broadly similar.

```
y1_design <- seq(min(y1), max(y1), length=npts_1)
x21_design <- runif(npts_1, min(y1), max(y1))
xn_2 <- cbind(y1_design, x21_design)
y2 <- vector(length=npts_1)
for (i in 1:npts_1){
  y2[i] <- test_func2(xn_2[i,1], xn_2[i,2])
}

F_mat2 <- matrix(rep(1, npts_1))
nugget_2 <- 1e-7
GP_2 <- mlegp(xn_2, y2)
b2 <- GP_2$beta
corr_mat2 <- corr_mat_setup(corrfunc_Gauss_Cpp, xn_2, b2,
        nugget_2)

y2_design <- seq(min(y2), max(y2), length=npts_1)
x31_design <- runif(npts_1, min(y2), max(y2))
xn_3 <- cbind(y2_design, x31_design)
y3 <- vector(length=npts_1)
for (i in 1:npts_1){
  y3[i] <- test_func3(xn_3[i], xn_3)
}

F_mat3 <- matrix(rep(1, npts_1))
nugget_3 <- 1e-7
GP_3 <- mlegp(xn_3, y3)
```

```
b3 <- GP_3$beta
corr_mat3 <- corr_mat_setup(corrfunc_Gauss_Cpp, xn_3, b3,
        nugget_3)
```

Since this chain is a function of $x_{1,1}$ only for a known set of functions, it can be visualised by plotting the linked emulator predictions and the true value of y_3 across the range of $x_{1,1}$, as in Figures 3.5 and 3.6. We therefore set up two functions to do this, one for the simulation-based linked emulator and one for the theoretical linked emulator, which differ based on how the upper and lower bounds of the credible interval for y_3 are calculated.

```
plotemutest_simu <- function(GP_sample, x_seq, true_y, xn, vn){

  mean <- apply(GP_sample, 1, mean)
  boundlow <- apply(GP_sample, 1, quantile, 0.025)
  boundup <- apply(GP_sample, 1, quantile, 0.975)

  plot(x_seq, true_y, type="l", ylim=c(min(boundlow, true_y),
        max(boundup, true_y)), xlab=vn[1], ylab=vn[2])
  lines(x_seq, mean, col="blue")
  lines(x_seq, boundlow, col="green")
  lines(x_seq, boundup, col="green")
}

plotemutest_theory <- function(means, vars, x_seq,
        true_y, xn, vn){

  sds <- sqrt(vars)
  boundlow <- means - 2 *sds
  boundup <- means + 2 * sds

  plot(x_seq, true_y, type="l", ylim=c(min(boundlow, true_y),
        max(boundup, true_y)), xlab=vn[1], ylab=vn[2])
  lines(x_seq, means, col="blue")
  lines(x_seq, boundlow, col="green")
  lines(x_seq, boundup, col="green")
}
```

We now need to set up the set of prediction points for the linked emulator. A set of 201 points are chosen in a sequence between 0 and 1 for the first input $x_{1,1}$, with random values drawn for $x_{2,1}$ and $x_{3,1}$. Since the true form of the chain is known, we can also calculate the actual value of y_3 at each of the prediction points.

```
npts_test <- 201
x1_pred <- seq(0, 1, length=npts_test)
x2_pred <- runif(npts_test, min(y1), max(y1))
x3_pred <- runif(npts_test, min(y2), max(y2))
true_y1 <- apply(as.matrix(x1_pred), 1, test_func1)
true_y2 <- apply(as.matrix(true_y1), 1, test_func2)
true_y3 <- apply(as.matrix(true_y2), 1, test_func3)
```

Inputs for the function `MM_pred_simu` are then set up as described in Section 6.2, with a sample size of 1000 repetitions at each prediction point.

```
predpts <- list(x1_pred, x2_pred, x3_pred)
designs <- list(xn_1, xn_2, xn_3)
b_vecs <- list(b1, b2, b3)
res_vecs <- list(y1, y2, y3)

F_mats <- list(F_mat1, F_mat2, F_mat3)
corr_mats <- list(corr_mat1, corr_mat2, corr_mat3)
GP_sampsize <- 1000
loc_vec <- c(1, 1)
```

We are now in a position to use the `MM_pred_simu` function to obtain a sample of 1000 values for y_3 from the simulation-based linked emulator at each prediction point. The plot generated is the same as in Figure 3.5, but with the red dots for the design points removed, as the experimental design procedure introduced in Algorithm 2 does not allow for the design points in $x_{1,1}$ to be associated with a specific value of y_3.

```
y3_samples_simu <- MM_pred_simu(GP_sampsize, predpts, loc_vec,
        designs, b_vecs, res_vecs, F_mats, corr_mats,
        cov_type='Gaussian')
```

The output of the simulation-based linked emulator can be visualised and compared to the true value of y_3 at the prediction points using the function we set up earlier for this purpose. We use the `expression` function to generate the correct variable names in the plot labels.

```
plotemutest_simu(y3_samples_simu, x1_pred, true_y3, xn_1,
       c(expression('x'['1,1']), expression('y'[3])))
```

The additional inputs required for the `MM_pred_theory` function are set up as follows.

```
beta0_1 <- mean(y1)
beta0_2 <- mean(y2)
beta0_3 <- mean(y3)
beta0s <- list(beta0_1, beta0_2, beta0_3)

sig2_1 <- var(y1)
sig2_2 <- var(y2)
sig2_3 <- var(y3)
sig2s <- list(sig2_1, sig2_2, sig2_3)

inv_corrmat1 <- chol2inv(chol(corr_mat1))
inv_corrmat2 <- chol2inv(chol(corr_mat2))
inv_corrmat3 <- chol2inv(chol(corr_mat3))
inv_corrmats <- list(inv_corrmat1, inv_corrmat2, inv_corrmat3)

nuggets <- list(nugget_1, nugget_2, nugget_3)
```

`MM_pred_theory` can now be used to generate the mean and variance of y_3 under the theoretical linked emulator at each prediction point. These are again visualised and compared to the true values using the relevant function defined earlier in this section. `MM_pred_theory` returns a list containing the means and variances of y_3, so these must be passed to the plotting function separately. The plot is that seen in Figure 3.6 with the red dots for the design points removed as described above.

```
y3_result_theory <- MM_pred_theory(predpts, designs, b_vecs,
        res_vecs, inv_corrmats, beta0s, sig2s, nuggets)
y3_means <- y3_result_theory$means
y3_vars <- y3_result_theory$vars

plotemutest_theory(linked_means, linked_vars, x1_pred, true_y3,
        xn_1, c(expression('x'['1,1']), expression('y'[3])))
```

We now move on to sensitivity analysis. As in Section 5.6, we assume a uniform distribution on each of the controllable inputs to the chain. A sample size of 10000 will be used. We require two sets of samples, which are generated using the in-built `runif` function.

```
SA_sampsize <- 10000
sample_x11_1 <- runif(SA_sampsize)
```

```
sample_x11_2 <- runif(SA_sampsize)
sample_x21_1 <- runif(SA_sampsize, min(y1), max(y1))
sample_x21_2 <- runif(SA_sampsize, min(y1), max(y1))
sample_x31_1 <- runif(SA_sampsize, min(y2), max(y2))
sample_x31_2 <- runif(SA_sampsize, min(y2), max(y2))
```

Given these samples, we can conduct a complete sensitivity analysis on the third model in the chain given the inputs y_2 and $x_{3,1}$ using the `SensFinal` function. The samples must be formatted into two lists for use in the function. `SensFinal` generates a main effects plot of each input, and returns the Sobol' indices for the main effects and second-order interaction term. In this example, we do not specify any plots of bounds on the main effects, nor a contour plot of the interactions, but these can easily be added if desired. The plot generated can be seen in Figure 5.15.

```
samples1 <- list(sample_x11_1, sample_x21_1, sample_x31_1)
samples2 <- list(sample_x11_2, sample_x21_2, sample_x31_2)

SensFinal(designs, b_vecs, res_vecs, inv_corrmats, beta0s,
        sig2s, nuggets, samples1, samples2,
        varnames=c(expression('y'['2']), expression('x'['3,1'])))
```

We are also able to generate a plot of the main effect of each directly controllable input to a two-model chain using the `Multimod_calcME` function. We therefore use this to generate such a plot for y_2 against $x_{1,1}$ and $x_{2,1}$. Only one sample for the controllable inputs is required, which must be structured as a matrix instead of a list.

```
sample_2mod <- cbind(sample_x11_1, sample_x21_1)

Multimod_calcME(sample_2mod, xn_1, xn_2, y1, y2, beta0_1,
        beta0_2, inv_corrmat1, inv_corrmat2, b1, b2, sig2_1,
        sig2_2, nugget_1, nugget_2,
        varnames=c(expression('x'['1,1']), expression('x'['2,1'])))
```

This code generates the plot seen in Figure 5.16, and returns the numerical values which make up the plot in case these are required for further use. At present, our code does not allow us to generate a main effects plot for each controllable input to a chain of three models or more, hence why the final output y_3 is not considered in this example.

Chapter 7

Application: casualty prediction from a CBR release

To demonstrate the methods introduced above, we consider a chain for CBR modelling. The chain is designed to predict the probability of casualty from a CBR release, and consists of two models. It can be viewed as a simplified version of the chain introduced in Figure 1.1. The second model in the chain is a dose-response model for the probability of casualty. This takes two inputs, one of which is dosage. This input is not of direct interest in research terms, since it is itself influenced by many other factors.

The first model in the chain deals with atmospheric dispersion. The model provides its output on a grid of points in the form of either the concentration of the contaminant at specified time intervals, or the average concentration per hour across the simulation period. The latter value multiplied by the number of seconds in the simulation period gives the dosage of the pollutant received by a person at each point on the grid over the simulation period. This model can therefore be used to generate the dosage input for the casualty model, leading to a chain of two models. The models themselves are described in detail in Sections 7.1 and 7.2 respectively.

7.1 Dispersion model

The first model in the chain deals with the dispersion of a contaminant such as a chemical or biological agent. The Hybrid Single Particle Lagrangian Integrated Trajectory Model (HYSPLIT) model was developed by the Air Resources Laboratory of the National Oceanic and Atmospheric Administration in the United States of America, and is available for download publicly. Details of the model are available in Stein et al. (2015). The model is relatively computationally intensive, and has a high overhead in terms of setting up new runs, so is one which benefits significantly from emulation.

The inputs to the HYSPLIT model consist of meteorological data on a geographical grid, and the physical and emission properties of the contaminant. The model has an extremely large number of inputs, many of which are either not suitable (due to being categorical instead of continuous) or not of interest for our purposes. It was therefore decided to focus only on varying three of the inputs, while keeping the others fixed. The three inputs chosen were release rate, release duration and release time. Ranges and units for these inputs are given in Table 7.1.

Input	Units	Lower bound	Upper bound
Release rate	units of contaminant / hour	0	2
Release duration	hours	0	12
Release time	minutes	0	360

Table 7.1: Ranges and units for the three inputs to the HYSPLIT model which are allowed to vary in our example

A single contaminant from a single release location was assumed. The simulation time was fixed at 12 hours. The release location is fixed at latitude 40 degrees north, longitude 90 degrees west, with an elevation of 10 metres. This is the default release location in the HYSPLIT program, and corresponds to a location in central Illinois, United States. The default meteorology file and default values for other inputs such as the half-life and diffusivity of the pollutant were also used.

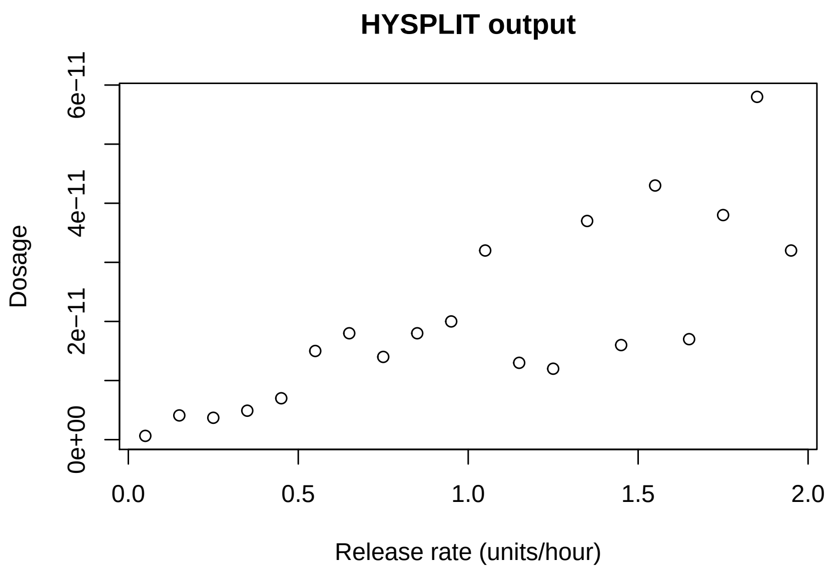

Figure 7.1: HYSPLIT output at the 20 design points against release rate

While the output is given on a grid, we are less interested in considering the effect across space and more concerned with the effect of the three inputs of interest. For this reason, a single point is chosen for analysis: the point with latitude 39.95 degrees north, longitude 90 degrees west. This is very close to the release location, differing only in being slightly further south. Since the wind direction specified in the meteorology file includes a strong northerly component, this location thus receives a relatively large dose of the pollutant over the course of the simulation compared to most on the measurement

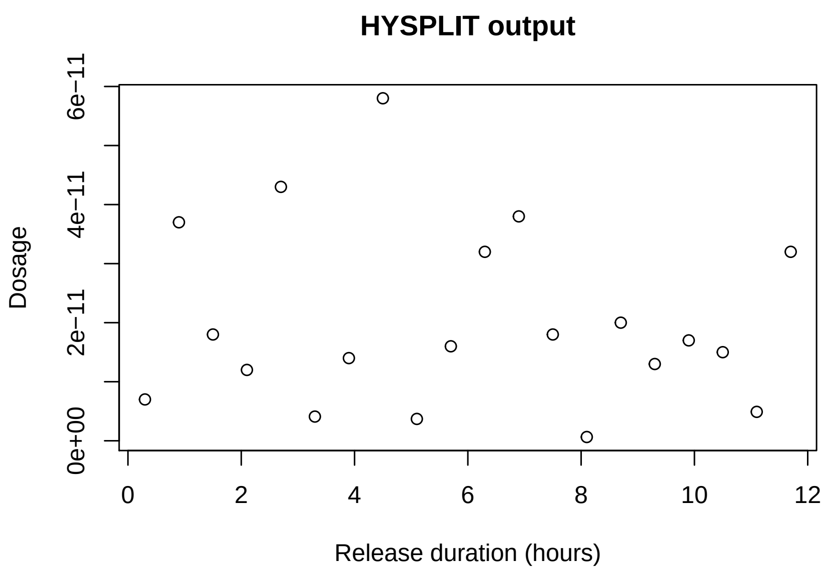

Figure 7.2: HYSPLIT output at the 20 design points against release duration

grid, although it is still possible for this dosage to be very small if the inputs of interest are chosen such that the absolute amount of pollutant released is low.

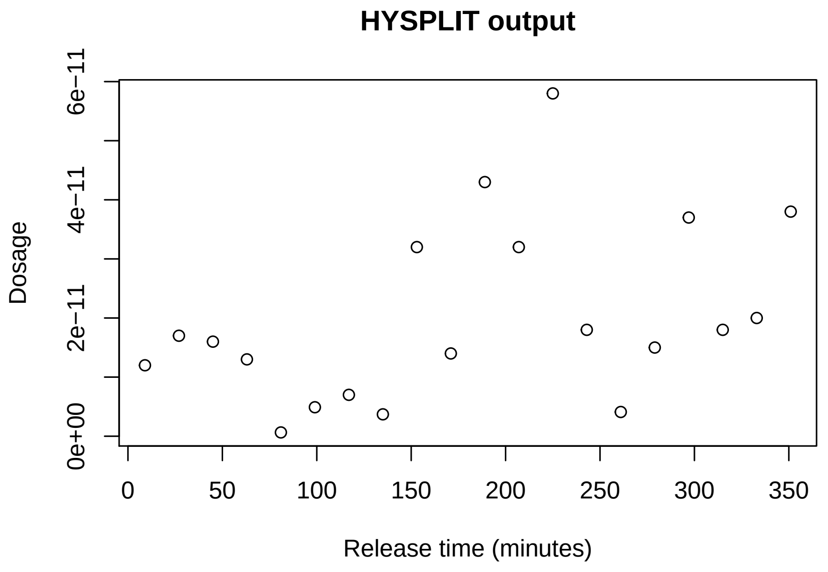

Figure 7.3: HYSPLIT output at the 20 design points against release time

All of the work which follows requires an emulator on the HYSPLIT model. A maximin Latin hypercube design with 20 points on the three inputs of interest (release rate, hours of emission and release time) is used to generate the data to which the emulator for the dosage output at the chosen spatial location is fitted. This is not a large design for a three-dimensional design space, so there is a relatively high level of uncertainty in prediction from the resulting emulator. The emulator is fitted using the `R` package '`DiceKriging`', with a Gaussian correlation function; the nugget is estimated from the data during the fitting process. The input variables are on widely differing ranges, so are rescaled to the $[0, 1]$ interval before the emulator is fitted.

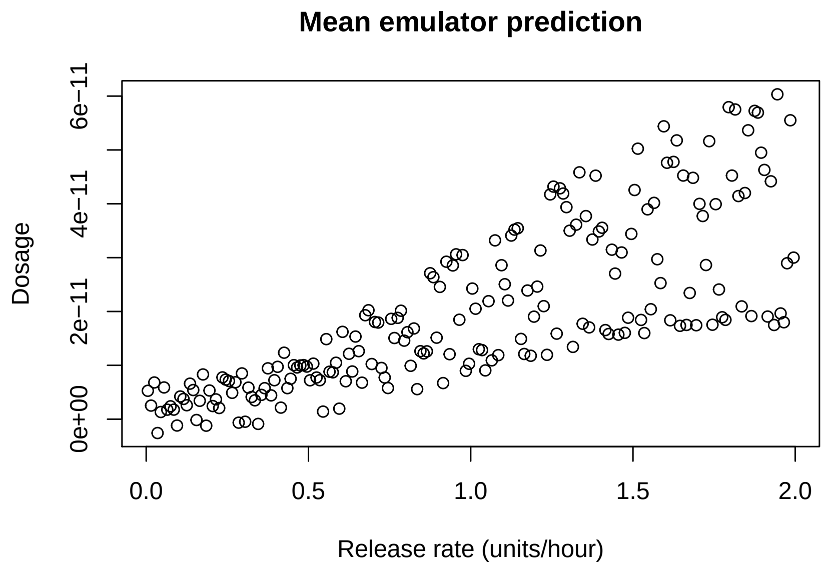

Figure 7.4: Mean emulator prediction for the HYSPLIT output at the 200 prediction points against release rate

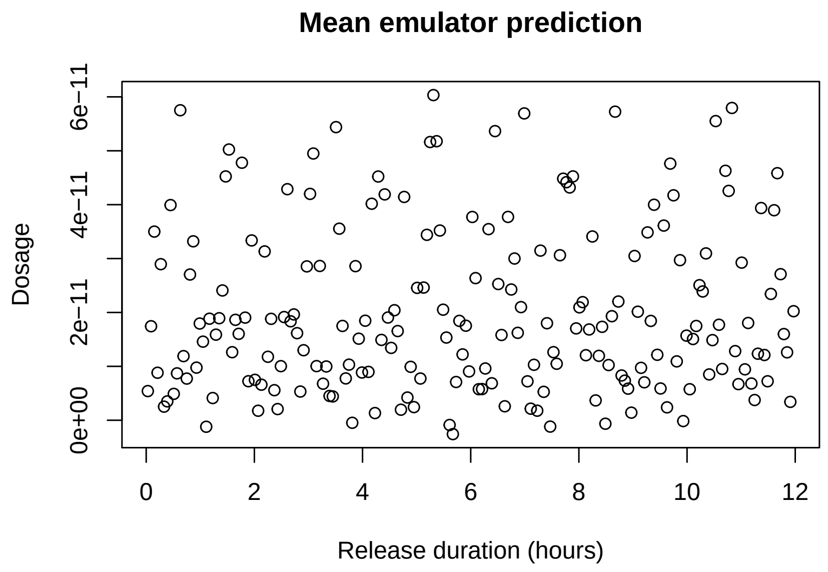

Figure 7.5: Mean emulator prediction for the HYSPLIT output at the 200 prediction points against release duration

To understand how the model behaves, it is useful to look at both the results of the simulator runs and the predictions made by the emulator for different input configurations. We can plot the true HYSPLIT output against the three inputs at the 20 points for which it is known. Figures 7.1, 7.2 and 7.3 suggest that there is a strongly increasing relationship between the release rate and dosage. A weaker relationship can be seen between release time and dosage, while there is no obvious pattern to the effect of release duration. These patterns are largely replicated, as we would expect, in the

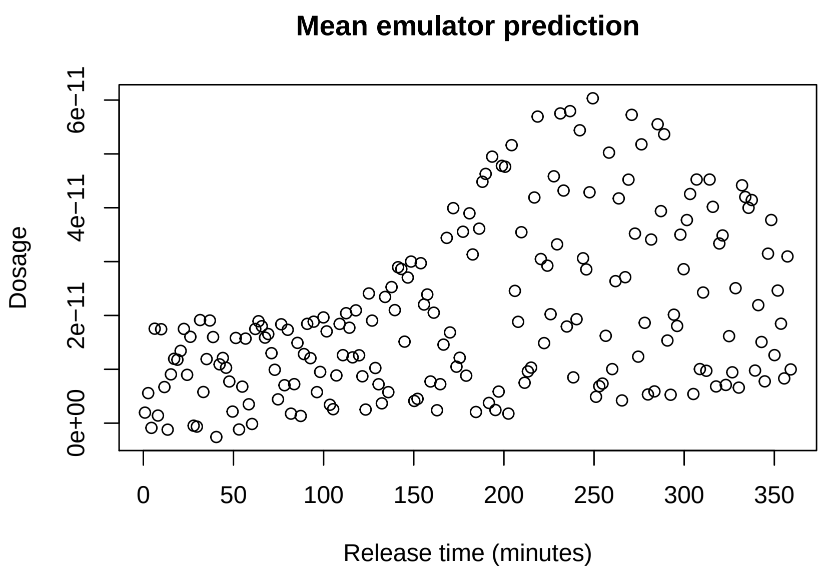

Figure 7.6: Mean emulator prediction for the HYSPLIT output at the 200 prediction points against release time

mean predictions of the emulator we fit to the data, displayed in Figures 7.4, 7.5 and 7.6. The observed trends in the model are not what may have been expected: typically, increasing the duration of release would cause the dosage to increase, while releasing the pollutant later may be associated with lower dosages. The unusual patterns seen here may be a result of the decision to focus on the dosage at a single geographical location close to the source of the release. It should also be noted that it is not easy to draw firm conclusions from these plots, as each individual scatter plot ignores the effect of changes in the other two variables.

One interesting feature of the predictions is that for seven of the 200 points, the mean emulator output is less than 0. This is of course not realistic, since a negative dosage of a pollutant cannot exist, and the HYSPLIT model does not return such values. Further analysis of these predictions shows that the variances associated with them are relatively large. It is also observed that all seven negative mean predictions occur when both the release rate and release time are low, a combination which is particularly difficult for the emulator to deal with since it leads to very low dosages. With only 20 design points for the computer experiment, it can be difficult for the emulator to make good predictions at extreme regions of the design space, especially when these regions are associated with extreme predictions. One way to deal with this would be to transform the output onto an unbounded scale before fitting an emulator. In this work, however, we shall continue to use the emulator described here for simplicity.

7.2 Casualty model

The second model concerns the probability of casualty arising from a given dose of a chemical or biological agent. It takes as inputs the dosage received by a person, together with two additional inputs D50 and slope which are properties of the contaminant. The output is the probability that the affected individual dies due to the agent.

The model equation for this particular dose response model is

$$P = 0.5\left\{1 + \text{erf}\left[\frac{s}{\sqrt{2}}\log_{10}\frac{x}{d}\right]\right\} \tag{7.1}$$

where erf is the error function, P is the probability of casualty, x is the dosage, d is the D50 and s is the slope. This model was suggested by Dstl to be indicative of what may be seen, and is termed a probit model for dose response. The use of probit models for casualty response to dosage has a long history, first appearing in Bliss (1934). Other dose response models are also available - Berkson (1944), for example, prefers the use of a logistic model, while Prentice (1976) introduces a four-parameter generalisation of both models.

In our work, the D50 and dosage are considered of interest to vary, while the slope is arbitrarily set to $s = 3$; this corresponds to a relatively gradual increase in the probability of casualty, with values of exactly 0 and 1 only at very extreme values of the other parameters. This was done to better demonstrate out methods and is not based on any real pollutants.

The model can be viewed as a simple reparameterisation of the standard probit regression equation, in which the constant term has been adjusted such that the role of the D50 input - a known or estimated property of the agent, corresponding to the dosage at which the probability of casualty is 0.5 - is made explicit. Equation (7.1) could be rewritten as

$$\begin{aligned} P &= 0.5\left\{1 + \text{erf}\left[\frac{s}{\sqrt{2}}(\log_{10} x - \log_{10} d)\right]\right\} \\ &= 0.5\{1 + \text{erf}[\beta_0 + \beta_1 \log_{10} x]\} \end{aligned}$$

where $\beta_1 = s/\sqrt{2}$ and $\beta_0 = \frac{s \log_{10} d}{\sqrt{2}}$. This is the standard form of a probit regression model.

7.3 Prediction from the chain

For this particular chain, there are several approaches that could be taken for prediction. The dose-response model is not computationally expensive to run, so Gaussian process emulation is not strictly required in order to make inferences about its behaviour. For the purposes of demonstrating our methodology, however, it will nonetheless be useful to emulate it anyway. In addition, while simulation-based methods lend themselves naturally to a chain of models in which only some will require emulation, the theoretical results derived in chapter 3 are specific to the case where all models in the chain are emulated. However, simulation using the true model provides a useful benchmark to compare our results again.

The following subsections will therefore consider four cases: a simulation in which only the first model is emulated; a composite emulator, where the output of the second model is modelled directly as a function of the controllable inputs to the chain only a linked emulator constructed by Monte Carlo integration in which both models are emulated; and a linked emulator constructed such that its theoretical mean and variance are known. To test prediction across the space of the four controllable inputs, a maximin Latin hypercube with 200 points on four dimensions is used. This prediction set will form the basis for comparisons between the different methods considered in this chapter.

7.3.1 Direct simulation on the dose-response model

Prediction in the first case is achieved by generating a set of 1000 predictions from the emulator for the HYSPLIT model for the values of release rate, release time and release duration at each prediction point, then feeding each of these outputs into the true simulator for dose response together with the value of D50 at the relevant prediction point. Since this method has no uncertainty arising from the second model, we would expect this to approximate the system better than one which has uncertainty in both stages of the chain, so this will be used as the benchmark to which the other methods are to be compared.

The plots of mean probability of casualty against the four inputs of interest (Figure 7.7) reveal a mixed picture. The most important input would appear to be D50, with low values strongly associated with high probabilities and high values with low ones. There is also a clear relationship between low release rates and low probability of casualty, although the effect across the rest of the range of this input is less clear. Release time shows a weaker relationship with casualty probability, while there is no obvious evidence of a pattern in the plot of probability against release duration. These results are consistent with what would be expected based on the preceding two sections. Dosage (not included in these plots as it is not a directly controllable input to the chain) and D50 should account for similar proportions of the variation in the probability of casualty. A low value of D50 means that only a small dosage is required to reach a probability of casualty of 0.5, while a high value means that a larger dosage is needed, so the pattern seen is this plot is to be expected.

The relationship between dosage and casualty probability is positive (since a high dosage is more likely to have a significant effect than a low one). In Section 7.1, we saw that increasing the release rate and release time is associated with an increase in dosage, so we would expect a positive relationship between these inputs and probability of casualty - but since dosage is a function of three inputs, the relationship should be less strong than between D50 and probability of casualty. The absence of a clear link

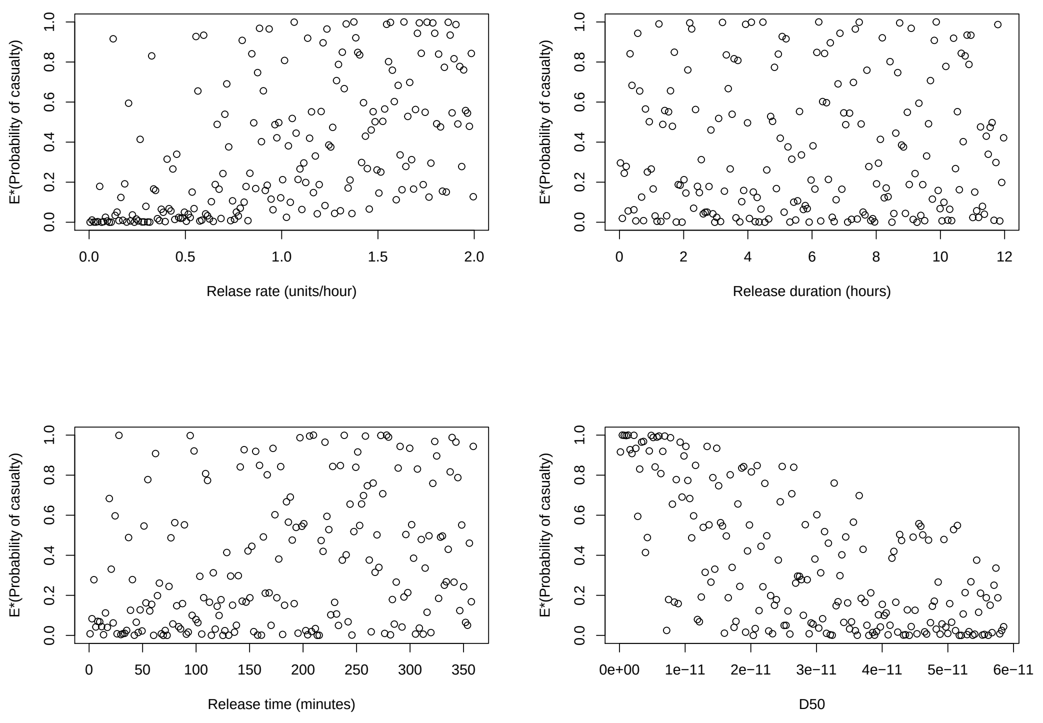

Figure 7.7: Mean prediction of probability of casualty against release rate (top left), release duration (top right), release time (bottom left) and D50 (bottom right) at the 200 prediction points - direct simulation on the dose-response model

between release duration and casualty probability is also consistent with what was seen in Section 7.1. Additionally, it is interesting to note that there are more predictions close to 0 than to 1.

It is also of interest to briefly investigate the variance of the predictions from the chain. Since all of the variation arises from the first model, it may be expected that the variance in our predictions would be related to the inputs to the first model in some way. However, the opposite is the case: as seen in Figure 7.8, the variance is generally very low across the predictions, with large values occurring almost entirely when D50 is low. The plots for the other inputs reveal no comparable trend in the location of the high-variance points. This may be because low values of D50 lead to potentially very high probabilities of casualty, which means that changes in the dosage obtained from the first emulator could have a large knock-on effect in the probability of casualty obtained from the second model.

7.3.2 Composite emulator

To build an emulator for the entire chain, a 20 point Latin hypercube design is again used, but this is now in four dimensions instead of three since D50 must be accounted for at this stage. The first model is run at the 20 configurations of its three inputs, and the obtained dosage is used as an input to the second model together with the D50

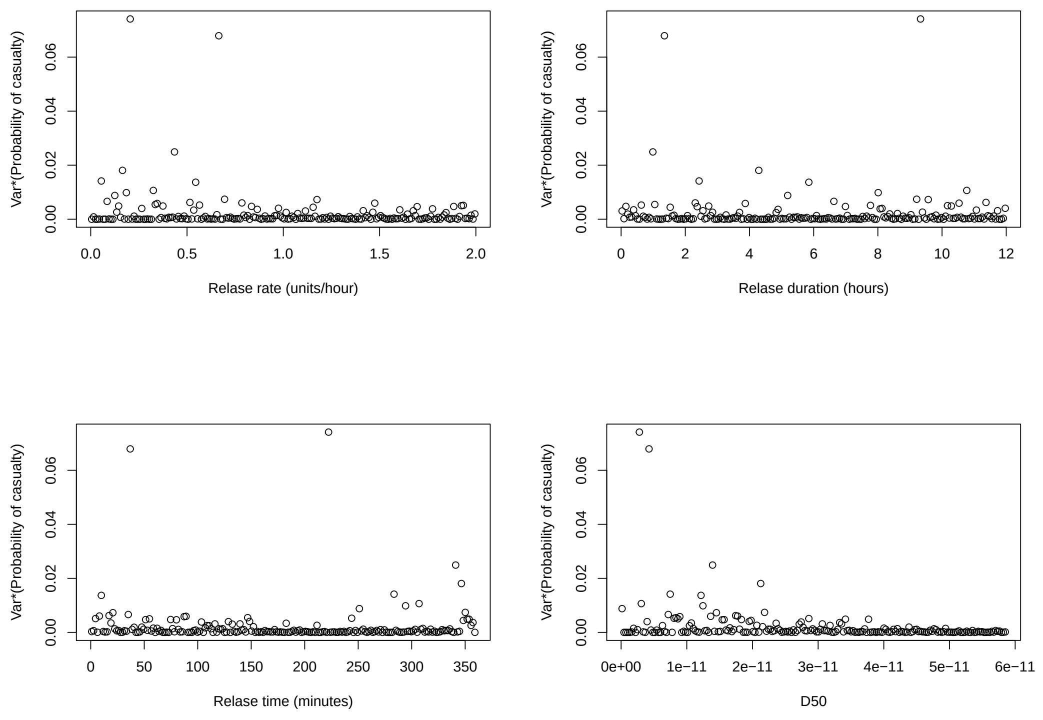

Figure 7.8: Prediction variance of probability of casualty against release rate (top left), release duration (top right), release time (bottom left) and D50 (bottom right) at the 200 prediction points - direct simulation on the dose-response model.

value specified by the design. An emulator is then built to approximate the probability of casualty directly from the four inputs. A Gaussian correlation function is used, with all parameters including the regression coefficients, process variance and nugget estimated from the data.

Predicting the output at the 200 prediction points gives a very different set of plots to those seen before. The composite emulator shows little link between release duration and probability of casualty, as the method based on direct simulation did, but disagrees with it on the effect of the other variables. While some evidence of the previously-observed patterns in release rate and release time are still present, much of their effect is lost, particularly at the lower end of release rate. The trend of decreasing probabilities with increasing D50 remains, but is bucked by an increased probability of casualty as the extreme upper end of its distribution; across most of the rest of the input range, predictions are too concentrated, with the emulator largely unable to predict outside of a relatively narrow range at similar values of D50 almost regardless of the other inputs. More of the mean predictions are close to 1 than in the previous case, and there are several mean predictions below 0 or above 1 (an artefact of emulating on a bounded scale instead of an unbounded one). In addition, the prediction variances are generally much larger than seen previously (although this is to be expected, since the second model is now a source of uncertainty), with little pattern in the variance relative to the

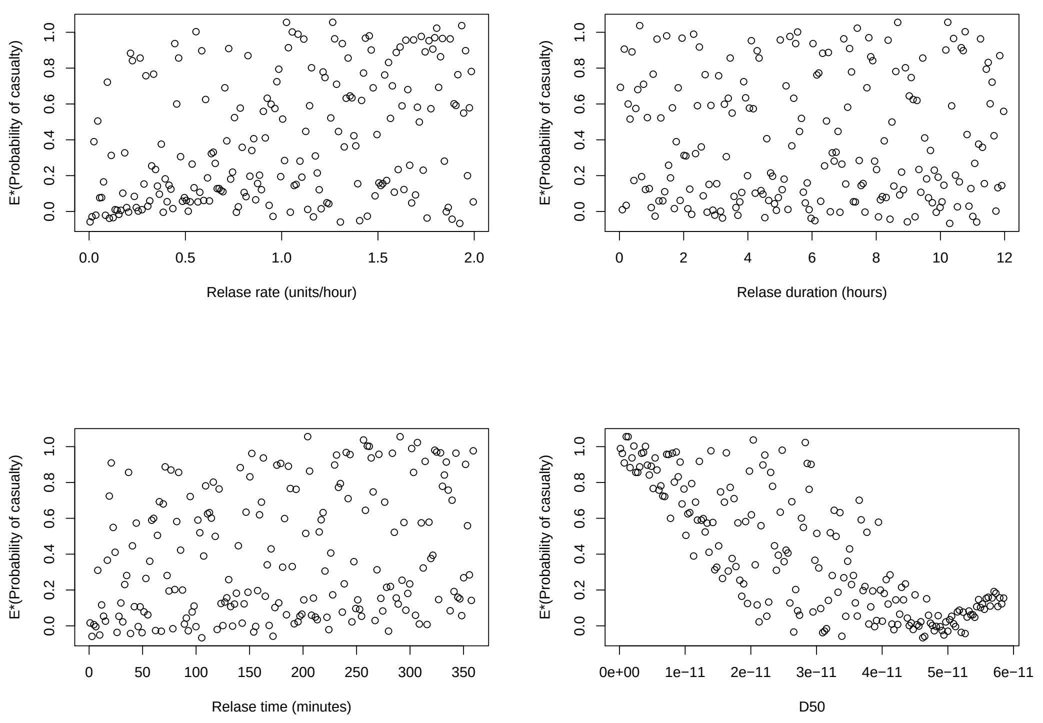

Figure 7.9: Mean prediction of probability of casualty against release rate (top left), release duration (top right), release time (bottom left) and D50 (bottom right) at the 200 prediction points - composite emulator

inputs.

Many of these problems are explained by the experimental design used to build the emulator. A 20-point design over a four-dimensional space is not especially large, and leaves sizable regions of the design space uncovered. This could be overcome by using a larger design, but this would require an increased number of runs of the complex HYSPLIT model. In addition, while the 20 design points are evenly spread on the four controllable inputs, they are not evenly spread on the dosage input to the casualty model. The increase in the predicted probability of casualty at the highest values of D50 is a result of these issues: the two design points with the largest values of D50 happen to occur at distinct configurations of the other three inputs which lead to higher-than-average dosages, which confuses the emulator into assuming that the higher probabilities observed are a result of the D50 input instead of the release rate and release time which actually explain the observations. Emulating the two models separately offers more flexibility in this respect, since the number of design points for the computer experiment on each model need not be the same.

7.3.3 Theoretical linked emulator

To construct the linked emulator, we require separate emulators for the two models in the chain. The HYSPLIT model is handled as before, using the 20 design points intro-

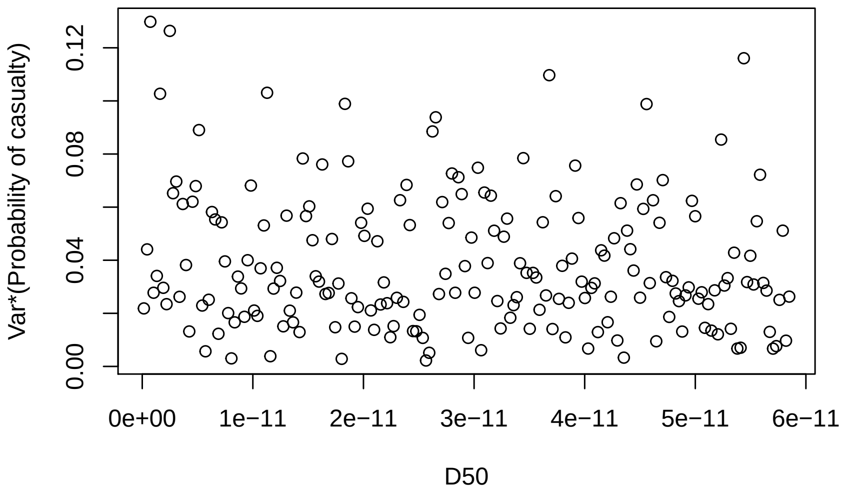

Figure 7.10: Prediction variance of probability of casualty against D50 - composite emulator

duced in Section 7.1. For the significantly cheaper casualty model, we take advantage of the split nature of the emulators to use a larger experimental design with 50 points. This is chosen using a two-dimensional maximin Latin hypercube, with the range on which dosage is taken to vary determined by the results of the 20 HYSPLIT model runs. This approach is preferable to using the results of the HYSPLIT model runs as design points for the reasons discussed in Chapter 4; the more complex sequential design ideas are not considered here, but could potentially be of use for future study on the same topic. A Gaussian correlation function is chosen for both emulators.

We first consider a linked emulator where the theoretical mean and variance can be determined. Plug-in estimates of the regression coefficients and process variances for each emulator are obtained using the `'DiceKriging'` package.

Plotting the probability of casualty against the four inputs (Figure 7.11) reveals a similar picture to that seen in Figure 7.7. The same inputs are deemed to be important, and their effects on the output is largely similar in nature. This suggests that the linked emulator has captured the features of the chain of models relatively well. There are however several predictions outside of the $[0, 1]$ range on which probability of casualty lies, which is a source of some concern, but these are fewer in number and generally lower in magnitude than under the composite emulator. These could in theory be dealt with using a transformation which takes the probability of casualty to the whole real line, but when this was attempted in practice, the result was an emulator which failed to accurately capture the behaviour of the casualty model.

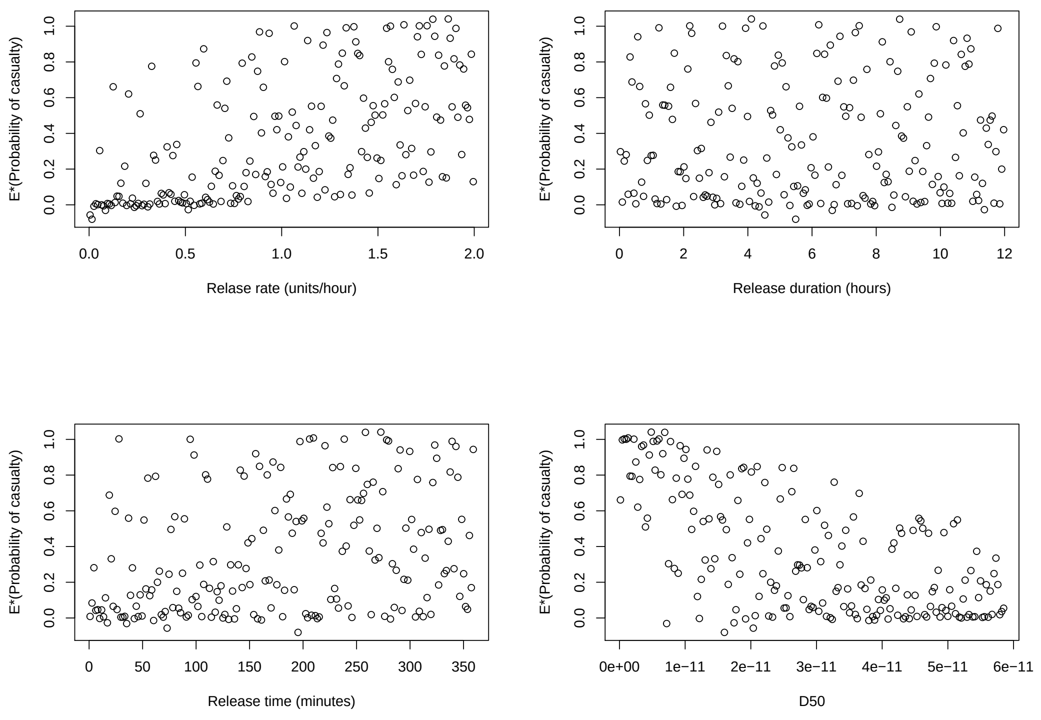

Figure 7.11: Mean prediction of probability of casualty against release rate (top left), release duration (top right), release time (bottom left) and D50 (bottom right) at the 200 prediction points - linked emulator, theoretical method

7.3.4 Simulation-based linked emulator

The simulation approach allows for more flexibility. We use this to integrate out the regression coefficients and process variances instead of using plug-in estimates, which allows us to better account for the uncertainty arising from their unknown status. The simulation itself proceeds as described in Section 3.2, with 1000 repetitions at each design point.

Again reviewing the plots of probability of casualty against the four inputs (Figure 7.12), we observe that the patterns are very similar to those seen in Figure 7.11 when the theoretical method was used. Integrating out the additional parameters of the two GP emulators does not appear to have changed the mean response significantly. This is as expected: the two methods use the same data and similar structures for the two emulators in the chain, so it would be highly concerning if they disagreed markedly. However, the additional uncertainty captured in the simulation method may lead to larger changes in the variance. This is investigated in Figure 7.13, where the variances of the two methods are plotted together against D50. First, it is important to note that the pattern in the variances is much more like that of the direct simulation method than the composite emulator, which further suggests that the linked emulator is capturing the behaviour of the chain well. Both linked emulator approaches give larger variances than direct simulation on the casualty model, which is again to be expected as the

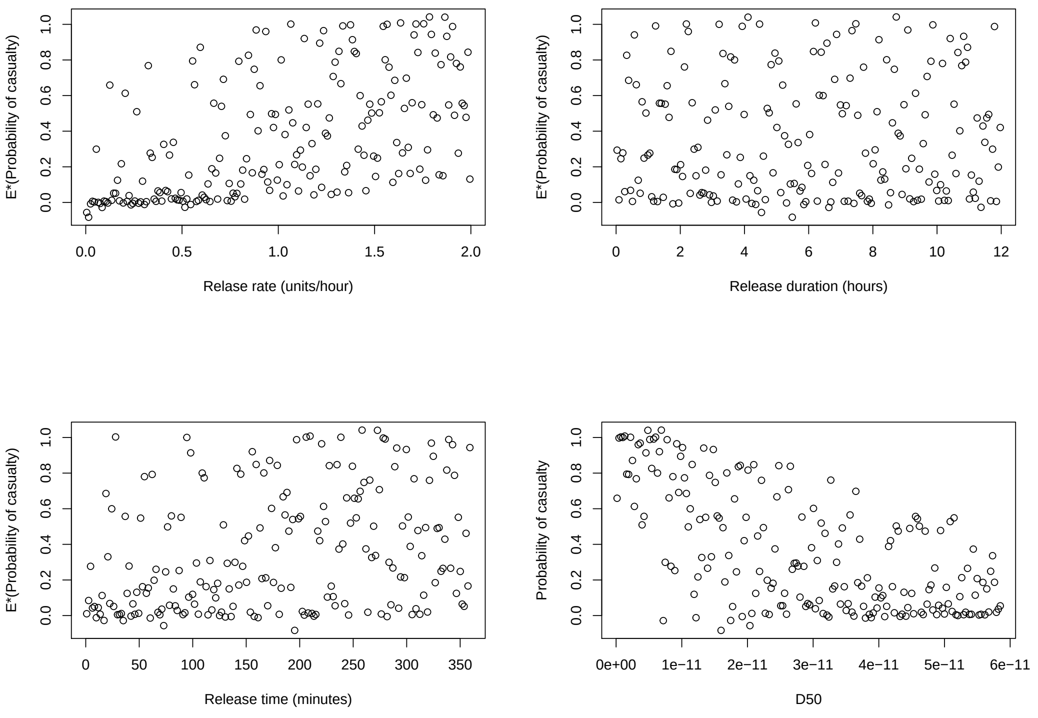

Figure 7.12: Mean prediction of probability of casualty against release rate (top left), release duration (top right), release time (bottom left) and D50 (bottom right) at the 200 prediction points - linked emulator, simulation method.

second emulation step introduces additional uncertainty into the predictions.

It is also clear from Figure 7.13 that the variances in the simulation method (plotted in red) are consistently slightly larger than those of the theoretical method (in black) at the same prediction points. The difference in variance is not uniform across the space: it affects some points more than others, and is generally larger where the variance is already relatively high; this may be a true reflection of an underlying pattern, or an artefact of Monte Carlo error in the simulation. The observed variances nonetheless remain low relative to the scale of the predictions themselves. It is clear that although considering for uncertainty in the GP parameters does increase prediction variance, the extra variability accounted for is not especially large in this case. It should however be noted that this method does not take account of uncertainty in the correlation parameters of the two GP emulators, which would have to be dealt with using a Markov chain Monte Carlo approach as discussed in Section 2.3.

7.4 Sensitivity analysis

Having concluded that the linked emulator provides a reasonable approximation to the chain, we now move on to sensitivity analysis. Both sensitivity analysis on the dose-response model (with respect to its inputs, dosage and D50) and sensitivity analysis

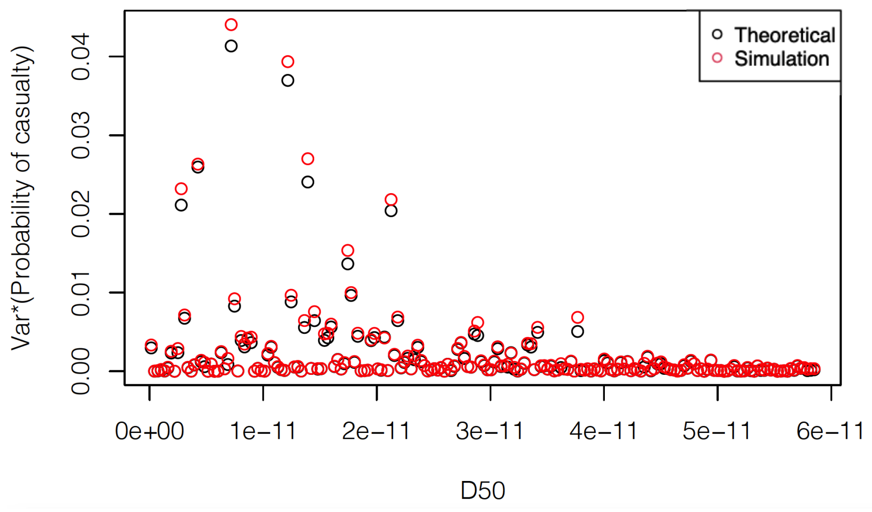

Figure 7.13: Prediction variance of probability of casualty against D50 - linked emulator, theoretical method (black) and simulation method (red)

for the entire chain will be considered.

Before any analysis can be conducted, we require distributions for the four controllable input variables. We have no particular information about the three inputs to the HYSPLIT model - release rate, release duration and release time - beyond the range on which each can occur. They are thus best handled using uniform distributions on their range. The distribution which best represents the behaviour of the D50 input is more debatable. Unlike the three inputs previously discussed, in which different configurations correspond to different types of release which may all be of interest, D50 is a property of the agent being released itself. Assuming the agent is the same in each, this has a single true value, but it may not be known exactly. A symmetrical distribution about the most likely value would thus appear sensible; the obvious choice is a truncated normal distribution with mean at the centre of the range of possible values for D50. The variance of the normal distribution could be chosen in several ways depending on how uncertain we are about the true value. The limiting case of the variance being effectively infinite on the input range corresponds to a uniform distribution for D50. In this work, we consider a uniform distribution; a normal distribution with finite variance would also be a reasonable choice, with the variance chosen based on discussions with a subject-matter expert.

First, we consider sensitivity analysis for the dose-response model given its inputs. The dosage input arises as the output of the HYSPLIT model, so the distribution of this input must be derived empirically by prediction from the GP emulator for the HYSPLIT model across a large set of input configurations from the chosen distributions on release rate, duration and time. The resulting Sobol' indices are 0.4576 for dosage, 0.4809 for D50 and 0.0615 for the interaction between them. This suggests that both inputs are of similar importance given the specified distributions on the controllable inputs. The corresponding main effects plot (Figure 7.14) shows that the effect of increasing dosage is to increase the probability of casualty on average, while the effect of increasing D50 is to reduce it on average; this is consistent with what might be expected based on the form of the dose-response model given in (7.1).

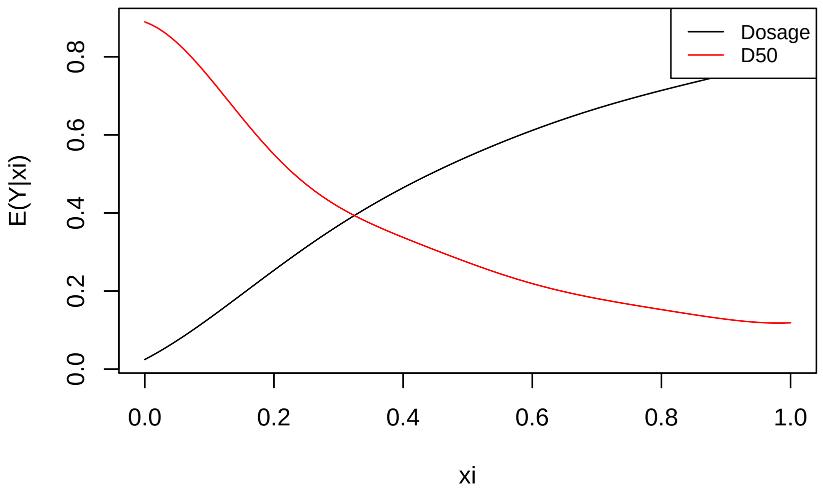

Figure 7.14: Posterior expectation of probability of casualty given the two inputs to the dose-response model.

To analyse the chain as a whole requires substantially more input configurations to be tested than the previous small-scale prediction studies, so execution speed is of the essence, which means the slower simulation-based methods are of less use here. The theoretical results from the linked emulator are therefore used as the basis for our analysis.

Figure 7.15. shows a (scaled) main effects plot for each controllable input to the chain. The inputs are scaled to the $[0, 1]$ interval so that they can be directly compared on the same plot; only the posterior expectation is considered. Each input is fixed at 50 equally-spaced values across its range, with 1000 random values drawn from the distribution of the unfixed variables, and the expectation of the linked emulator means across these 1000 runs taken to obtain an estimate of the posterior expectation of the probability of casualty given the fixed input.

Having removed the confounding effects of the other variables, it is now possible to see the patterns in the behaviour of the probability of casualty more clearly. There is a strongly negative relationship between D50 and probability of casualty, with the rate of decrease slowing as the input becomes larger. Increasing release rate is associated with increased probability of casualty, but the relationship is highly non-linear, with a slower rate of increase at the two extremes of its range. Release time has a small but

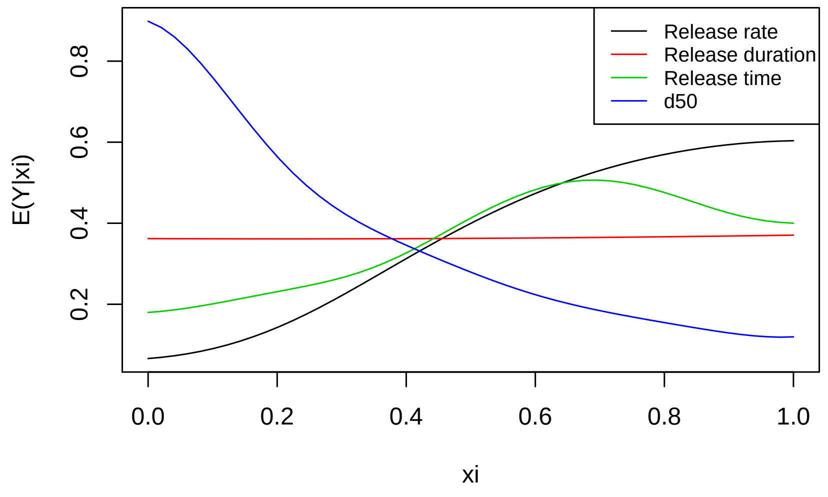

Figure 7.15: Posterior expectation of probability of casualty given each input to the linked emulator.

noticeable effect, which is initially positive but later negative in nature as the variable is increased across its range. Release duration has little effect: fixing it to any value provides no substantial information on the output of the chain of models. Despite the previously-observed individual predictions outside of the range $[0, 1]$, there is no value of any input which gives a posterior expectation of the expected value of probability of casualty outside of this range.

Ideally, we would of course like to consider sensitivity analysis for the full chain further, by constructing variance bounds for the main effects and calculating Sobol' indices for the main effects and two-way interactions. As discussed in Chapter 5, this is not currently possible. For this specific example, further analysis of the chain could be conducted by simulation by making use of the emulator for the HYSPLIT model and using direct simulation on the dose-response model. This is not directly relevant to the framework we have constructed in this thesis, however, so was not considered here.

7.5 Conclusions

This chapter uses the methods developed throughout this thesis to make predictions from a chain of models for a real-life scenario. Being a CBR chain, it is directly relevant to the research goals set out in Chapter 1. Taking advantage of the simplicity of the second model allowed our predictions to be compared directly to the output of a simulation where only one model is emulated, which should better reflect the true nature of the chain. The results confirmed both the consistency of the two linked emulator approaches, and that both outperform the composite emulator for this particular chain, thus validating the linked emulator approach. The results also serve as a comparison between the two linked emulator strategies, demonstrating both the speed and exact nature of the theoretical method and the flexibility and more robust uncertainty estimates arising from the Monte Carlo method.

Sensitivity analysis demonstrates that the D50 input of significant importance - effectively equal to that of the dosage, and greater than that of any individual input to the HYSPLIT model. One conclusion from these results, since the inputs to the dispersion model have a range of interest on which they can vary while D50 has a single true (but unknown) value, is that learning more about D50 would have a significant effect in reducing the uncertainty in predictions from the chain of models.

We should note that this chain of models is far from ideal for the purposes of making new inferences using our methodology. The chain is only two models long, and could thus have already been analysed using the pre-existing methods set out in Section 3.3; a longer chain would offer more potential for new results that could not already have been obtained. More importantly, the second model is too simple to require emulation at all. The chain is also incomplete when compared to the diagram in Figure 1.1, with many more modelling aspects which could be considered. This is discussed further in Chapter 9.

Chapter 8

Simulation study

8.1 Introduction

In Chapter 7, we presented an analysis of the linked emulator method when applied to a real-world problem. Our results demonstrate that both of the linked emulator approaches considered in this thesis can make better predictions than the simpler method of a composite emulator. However, it is difficult to draw conclusions from the analysis of a single chain of models, especially a chain in which the second function is very smooth and does not present a significant challenge to emulate.

A further aspect of linked emulator methodology which we have not yet explored in detail is the difference between the theoretical and simulation approaches. In both the dispersion-dose response chain in Chapter 7 and the one-dimensional synthetic chains in Chapters 3 and 4, the theoretical and simulation methods produced largely similar results. If this were true in general, there would be no need for the simulation method to be used, as it is significantly slower than the theoretical approach. But in all of these examples, the simulators being emulated were sufficiently smooth that a Gaussian correlation function could be used to build the emulators. The Gaussian correlation function produces infinitely differentiable sample paths, so there are some functions which cannot be emulated well using this correlation function. Stein (1999), pp.30-31 and 69-70, advises against using the Gaussian correlation function when modelling any physical process, citing both theoretical concerns and evidence in real examples of predictions which would be highly implausible in the context of the process being modelled. While computer experimentation does not involve modelling physical processes directly, a simulator that approximates a physical process should have similar behaviour to it. It is therefore of interest to investigate the behaviour of the two classes of linked emulators when a different correlation function may be more appropriate.

The comparatively poor performance of a composite emulator relative to its linked emulator counterparts in previous examples is also worthy of further consideration. A similar result was previously observed for a chain of simple synthetic functions by Kyzyurova et al. (2018), and it is especially noteworthy in the real example in Chapter

7. It is however unclear if the failure of the composite emulator is specific to the types of chain considered so far, or a more widespread phenomenon. This is especially relevant to chains containing models which a Gaussian correlation function is not appropriate, as the simulation method for linked emulation is very slow in comparison to simply fitting a single emulator.

8.2 Simulation setup

These questions can be investigated by constructing a chain from models which require a less smooth correlation function but which also consist of known, easily computable functions. A set of simple functions with these properties are those with discontinuous derivatives, notably the absolute value function. We will therefore consider a chain of two models defined by the following relationships:

$$y_1 = [1 - (|x_1| * |x_2|^{3/2})^a]^3 \, ;$$

$$y_2 = \cos[(1 - 2 * |0.8 - y_1|^b)^3] \, ,$$

where $-1 \leq x_1 \leq 1$ and $-1 \leq x_2 \leq 1$. The values a and b are tunable parameters.. For the purposes of this study, we will consider five values each for these parameters: $a = \{2, 2.5, 3, 3.5, 4\}$ and $b = \{1, 1.2, 1.4, 1.6, 1.8\}$. These two sets combine to give 25 chains of models to investigate.

For each of these 25 chains, we build two linked emulators - one each using the theoretical and simulation methods - and a composite emulator. The composite emulators use a constant regression term, and a Matérn correlation function as defined in equation (2.3). The smoothness parameter of the Matérn correlation function is set to $\omega = 3/2$, with the remaining correlation parameters and the nugget estimated from the data. This choice of correlation function and smoothness parameter implies a Gaussian process prior which is differentiable only once, so is likely to be appropriate for a chain made up of less smooth simulators.

An experimental design is required for the directly controllable inputs x_1 and x_2 for the composite emulator. The same design can be used for all values of a and b; we choose 15 design points using a maximin Latin hypercube design, which thus requires the two simulators to be run 15 times each for a total computational cost of 30 simulator runs. The design points are given in Table 8.1.

In the interests of fairness, the two linked emulators for each chain share an experimental design and thus differ only in how they are fitted to the training data. However, since the first model in the chain is a function of two inputs x_1 and x_2, while the second is a function of just one input y_1, the linked emulator would benefit from allocating more simulator runs to the first model than the second. We thus allocate 20

x_1	x_2
-0.9333	0.2667
-0.8	-0.2667
-0.6667	0.6667
-0.5333	-0.6667
-0.4	0
-0.2667	0.5333
-0.1333	-0.9333
0	-0.4
0.1333	0.1333
0.2667	0.8
0.4	-0.8
0.5333	-0.1333
0.6667	0.4
0.8	0.9333
0.9333	-0.5333

Table 8.1: Experimental design for the complete chain of models for the composite emulator

design points to the first model and 10 to the second. Since both simulators have very similar computational cost, the total computational cost of the 30 simulator runs per linked emulator is virtually identical to that of the 30 simulator runs required for each composite emulator.

The design points for x_1 and x_2 are independent of a and b and can therefore be the same for all 25 chains. A 20-point maximin Latin hypercube design is used, with the design points presented in Table 8.2. The 10 design points for y_1 in the second model are chosen by following the procedure in Algorithm 2, and thus depend on the output of the simulator runs for the first model. The second simulator is univariate so the design points are equally spaced on the five different implied ranges of y_1, which are dependent on the value of a so cannot be made identical for every chain.

The linked emulator for the theoretical method is composed of two individual emulators, one for each simulator, with constant regression terms and Gaussian correlation functions as defined in equation (2.2). The regression coefficients, process variances, correlation parameters and nuggets are estimated from the data. The linked emulator for the simulation method is composed of two emulators with constant regression terms and Matérn correlation functions where $\omega = 3/2$, as for the composite emulators. The correlation parameters and nuggets are estimated from the data, using the likelihood marginal to the regression coefficients and process variances.

For an additional comparison to a non-emulation method, we shall also consider a linear regression model for y_2 against x_1 and x_2:

$$y_2 = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon \,,$$

x_1	x_2
-0.95	0.05
-0.85	0.65
-0.75	-0.45
-0.65	-0.95
-0.55	0.25
-0.45	0.75
-0.35	-0.25
-0.25	-0.75
-0.15	0.15
-0.05	0.55
0.05	0.95
0.15	-0.55
0.25	-0.05
0.35	0.45
0.45	-0.85
0.55	-0.35
0.65	0.85
0.75	0.35
0.85	-0.65
0.95	-0.15

Table 8.2: Experimental design for the first model in the chain for the linked emulator

where ϵ is an error term and the coefficients $\beta_0, \beta_1, \beta_2$ are estimated using maximum likelihood estimation.

To test the predictive capabilities of the four methods, we require a set of prediction points. We choose 1000 points across the space of x_1, x_2 from a maximin Latin hypercube. The first simulator was run at each of these points, and the resulting value of y_1 fed into the second simulator to obtain the true output of the chain. These true values are used as a comparison set for the predictions made by the composite emulator, the two linked emulators and the linear regression model at the same 1000 points. We consider two measures of predictive ability for the three methods. The first is the root mean squared error (RMSE) of the mean prediction from the regression and emulation methods versus the true value across the 1000 prediction points. Let η^* be the estimate of the true chain of models η, and let $\mathbf{x}_{1,i}$ be the ith prediction point. The RMSE of η^* is defined as

$$RMSE(\eta^*) = \sqrt{\frac{1}{n}\sum_{i=1}^{1000}[\eta^*(\mathbf{x}_{1,i}) - \eta(\mathbf{x}_{1,i})]^2}\,.$$

This is chosen instead of, say, the mean absolute error as it has the property of penalising large errors at individual prediction points more heavily, which is important as we do not wish the worst predictions made to be very large. The second measure is based on coverage. The composite emulator, both linked emulators and the linear regression model can generate nominal 95% prediction intervals for the output of the chain at any input set. A useful measure of the accuracy of the uncertainty estimates of the four methods is to compare this to the true percentage of the prediction set for which the calculated interval includes the true output. In particular, if for any method this is consistently below 95 or occasionally dramatically lower, the method is understating the uncertainty in its predictions.

8.3 Results

The resulting RMSE and coverage for each of the three methods across the 25 chains of models are presented in Table 8.3. For presentational reasons, we define the following abbreviations: Simu, the simulation-based linked emulator; Theory, the theoretical linked emulator; Comp, the composite emulator; Linear, the linear regression model.

a	b	RMSE				Coverage (%)			
		Simu	Theory	Comp	Linear	Simu	Theory	Comp	Linear
2	1	0.0804	0.0688	0.0980	0.1073	95.7	72.3	91.7	95.2
2	1.2	0.0833	0.0670	0.1126	0.1193	96.3	65.7	90.3	96.8
2	1.4	0.0805	0.0628	0.1186	0.1229	96.5	57.0	91.0	98.7
2	1.6	0.0781	0.0611	0.1199	0.1238	97.8	45.3	99.1	98.1
2	1.8	0.0779	0.0609	0.1207	0.1249	98.4	43.4	99.4	92.6
2.5	1	0.0896	0.0870	0.0936	0.1179	99.0	90.7	84.7	99.1
2.5	1.2	0.0955	0.0917	0.1057	0.1304	97.8	86.1	84.4	99.5
2.5	1.4	0.0946	0.0933	0.1089	0.1314	95.5	81.5	85.8	100
2.5	1.6	0.0937	0.0941	0.1186	0.1275	93.6	79.2	75.0	97.3
2.5	1.8	0.0941	0.0946	0.1246	0.1230	93.0	76.4	71.7	92.6
3	1	0.0893	0.0910	0.0879	0.0944	95.6	87.8	59.8	95.9
3	1.2	0.0993	0.1027	0.0987	0.1076	92.6	85.2	58.1	97.4
3	1.4	0.1012	0.1067	0.1028	0.1114	92.5	82.0	57.4	98.9
3	1.6	0.1000	0.1063	0.1025	0.1101	92.0	78.9	56.3	96.9
3	1.8	0.0990	0.1042	0.0998	0.1072	92.3	78.1	56.7	93.4
3.5	1	0.0875	0.0894	0.0836	0.0789	96.8	91.9	65.5	97.6
3.5	1.2	0.0968	0.1010	0.0984	0.0886	95.1	87.1	62.0	98.1
3.5	1.4	0.0979	0.1040	0.1026	0.0921	93.5	82.8	61.6	98.7
3.5	1.6	0.0959	0.1022	0.1005	0.0920	93.2	80.7	62.5	96.5
3.5	1.8	0.0939	0.0989	0.0960	0.0904	93.0	79.5	63.9	94.6
4	1	0.0794	0.0824	0.1103	0.0706	95.9	91.0	74.5	95.6
4	1.2	0.0878	0.0915	0.1190	0.0789	94.8	87.6	73.0	95.4
4	1.4	0.0893	0.0942	0.1164	0.0809	93.6	83.7	70.3	95.6
4	1.6	0.0879	0.0932	0.1082	0.0797	93.0	82.2	68.8	92.1
4	1.8	0.0857	0.0905	0.0987	0.0777	93.0	80.6	67.0	91.3

Table 8.3: RMSE and coverage for the simulation (S) and theoretical (T) linked emulators, for the composite emulator (C), and for the linear regression model (L).

There are several points of interest in these results. Firstly, the performance of the composite emulator is comparatively poor in most cases: across the 25 values of a and b given here, the RMSE for the composite emulator is the lowest of the three emulation-based methods in only two cases, compared to 15 cases where it is the highest. Only when $a = 3$ or $a = 3.5$ is the composite emulator competitive. There is also a connection to value of b: the composite emulator predicts better when b is small, except when a is large. The coverage of the nominal 95% prediction intervals for the composite emulator are also of concern, falling below 95% in all but two cases, and consistently very low whenever a is not equal to 1. When $a = 3$, which produces the relatively best RMSEs for the composite emulator, all five actual coverages are unacceptably low at below 60%.

The linear model varies strongly with the value of a in its predictive accuracy. For $a \leq 3$, its RMSE is consistently the largest of the four methods, suggesting a model which approximates the true chain poorly. When the value of a is 3.5 or 4, however, the situation is reversed: the linear model now outperforms all three emulation-based approaches. Despite this improved performance for a subset of the chains, it is clear that a linear model is insufficient to make good predictions in general. The coverage achieved by the linear model is strong for every chain, consistently exceeding 90% and exceeding 95% in 19 of the 25 cases. If anything, this suggests the prediction intervals provided are in fact too wide, although it is noteworthy that the coverage falls below 95% whenever $b = 1.8$.

The relative performances of the two linked emulators in terms of RMSE is varied, with 17 cases in which the simulation method produces the lower RMSE and 8 in which the theoretical method is the more accurate. Again, these are closely linked to the value of a. When $a = 2$, the theoretical method consistently produces the lower RMSE; for $a = 2.5$, there is little to separate the two linked emulators; while for $a \geq 3$, the RMSE of the simulation method is consistently lower. The main difference between the two methods is the choice of correlation function, so this suggests that low values of a produce a smoother simulator which is better suited to the Gaussian correlation function, while larger values of a lead to simulators which require a rougher correlation function like the Matérn. The composite emulator performs poorly when a is large despite also using a Matérn correlation function, which again indicates that the linked emulator approach is preferable.

However, the two types of linked emulator are vastly different in terms of the coverages of their nominal 95% prediction intervals. The coverages seen when the simulation method is used are largely consistent with what would be expected from a genuine 95% prediction interval: 12 are greater than and 13 less than 95%, with no value lower than 90%. The coverage is typically somewhat lower when both a and b are large, but the effect on the coverage of varying these parameters is not as pronounced as for the composite emulator.

The same cannot be said of the linked emulator constructed using the theoretical method. Instead, all 25 actual coverages are below 95%, suggesting a consistent problem of underestimating the uncertainty in the predictions made. The coverage is consistently poorer when b is large or when $a = 2$, and falls below 50% when $a = 2, b = 1.6$ and when $a = 2, b = 1.8$. This is particularly surprising as, for these two parameter configurations, the theoretical linked emulator produced not only the lowest RMSEs of the four approaches, but the lowest RMSEs of any method across any of the 25 chains. While the theoretical linked emulator makes comparatively very accurate mean predictions in these cases, there is a substantially greater variance associated with prediction than the method would suggest.

As discussed in Chapter 3, the probability distribution of the prediction from a theoretical linked emulator given the inputs to the chain is approximated by a normal distribution with the variance calculated using equation 3.8. The consistently low coverages would thus suggest that this equation in fact produces an underestimate of the true variance. This is not entirely surprising: the theoretical linked emulator uses plug-in estimation for the regression coefficients, process variance and correlation parameters, ignoring a potential source of predictive uncertainty. The simulation-based linked emulator also uses plug-in estimation for the correlation parameters, so its impressive performance in terms of actual coverage implies that - at least when Matérn correlation function is used - the uncertainty in the correlation parameters contributes little to the predictive uncertainty for this chain of emulators.

This simulation study of 25 chains of simulators demonstrates several noteworthy points about the different methods considered in this thesis. We observe that the composite emulator generally performs poorly compared to both classes of linked emulator. There are some chains for which a theoretical linked emulator provides good approximations, and others for which it is not a viable choice. The simulation method offers greater robustness, delivering reasonable predictions and trustworthy 95% prediction intervals across all of the chains. The theoretical method shows signs of consistently underestimating the variance in its predictions and thus providing unreasonably narrow prediction intervals. The much simpler linear regression model can perform well for some chains with more linear relationships between the input and output, but performs poorly in the majority of cases.

The simulation method however has a clear disadvantage in speed. Benchmarking the computational time required to make predictions at the 1000 prediction points for each of the three methods, we found that the composite emulator was the fastest. This is to be expected, as fitting and making predictions from a single emulator is a swift process, and while the theoretical linked emulator is substantially slower, it is clear from the consistently poor performance of the composite emulator that the speed advantage alone does not make this method worth pursuing. However, the simulation-based linked emulator is around 50 times slower than the theoretical linked emulator. The additional computational resources required mean that the performance advantages offered by the simulation method would need to be extremely large to be worth pursuing, and for chains with a large number of models or inputs, the method may not be computationally feasible at all.

For this reason, when utilising a linked emulator for a real problem, we would choose the theoretical method whenever it could be expected to give reasonable results. But identifying which chains of models can be approximated appropriately by a theoretical linked emulator is a difficult problem. It is not usually possible to know in advance whether the models in the chain are smooth enough for a Gaussian correlation function to perform well. One possibility is to fit GP emulators with both Gaussian and Matérn correlation functions to each model in the chain from the same simulator runs, and use the diagnostics for GP emulator performance discussed by Bastos and O'Hagan (2009) to assess if these emulators are appropriate before linking them together. It may also be possible to narrow the performance gap between the two linked emulator methods by improving the speed of the simulation method, for example by writing more efficient code, although its dependence on Monte Carlo approximation means it will always be the slower of the two approaches.

Perhaps the most important conclusion from this study is that the 95% prediction interval of a theoretical linked emulator should not be considered entirely trustworthy. In all 25 chains considered here, the actual coverage was well below 95, with the lowest coverages coming when the mean predictions were on average closest to the true output of the chain. This could be improved by integrating out the regression coefficients of the individual emulators instead of using plug-in estimation, although it would not be possible to integrate out the process variance as the resulting distribution on the output of the earlier models in the chain would no longer be normal. Finally, while the much less complex linear regression model is unable to match the performance of the linked emulator methods in general, its comparatively strong performance for certain chains highlights the potential benefits of using emulators with linear instead of constant regression terms for each model in the chain.

Chapter 9

Conclusions and future work

9.1 Conclusions

In this thesis, we considered approaches towards the analysis of chains of complex computational models using Gaussian process emulation. We have presented a variety of methods by which such chains can be handled. We reaffirm existing results that suggest that a linked emulator (in which the models in the chain are emulated separately, and the uncertainty passed between them) is preferable to a composite emulator (in which a single emulator is built for the chain of models as a whole), and present two methods for analysis from such chains - a flexible approach involving simulation and Monte Carlo integration, and a faster theoretical approximation for a specific class of emulators. Both methods were generalised from a simple two-model chain to a longer chain, the latter using an approximation to a probability distribution which is not available in closed form. A simulation study was conducted to investigate the differences between the two approaches to linked emulation, demonstrating that the simulation method is more robust if the models in the chain are less smooth than ideal for an emulator with a Gaussian correlation function. The advantages of a linked emulator approach over the much simpler technique of linear regression was also demonstrated, with the caveat that the occasionally very good performance of linear regression provides a strong argument for the use of emulators which include a linear component.

We have also highlighted the related problems of experimental design for chains of models. We focus on experimental design for models at the second and later steps in the chain, illustrating why this is both an important and a difficult problem, and present a simple algorithm for single-stage design which nonetheless offers significant benefits over more naive approaches. Some thought was also given to sequential design for chains of models, both in terms of why this may be beneficial and to potentially productive strategies to accomplish it.

Sensitivity analysis for a chain of models was another area of focus. We developed an algorithm for sensitivity analysis for the final model in the chain in terms of its own inputs. We also worked towards sensitivity analysis for the final output of the chain in terms of the directly controllable inputs to any model in the chain, with a theoretical result presented for the posterior expectation of the main effects and interactions of an input or set of input.

Our work was tested on models supplied by Dstl, which form a simplified chain demonstrating some of the principles of casualty modelling from a CBR release. Although this example is imperfect - a two-model chain with a relatively simple second model - it nonetheless highlights the potential applications of our research. In addition, we were able to demonstrate fast predictions from the chain of models for different input configurations, and to draw new conclusions about the relative importance of the inputs to the model output.

There are many avenues of relevant future research that could be considered. These are discussed in detail below.

9.2 Future work: experimental design

One of the most promising fields for further research would appear to be experimental design for chains of computational models. This is a broad and complex topic with significant potential for further research. Our method for single-stage design for chains of emulators, presented in Section 4.2, effectively reduces the problem to one which has already been solved. As discussed in the same section, however, this is not perhaps the most rewarding approach to the design problem for chains of models.

Even here, however, there are open questions. Our method uses the results of the simulator runs for model 1 to construct a design space for y_1 in the second emulator. This is potentially problematic if the simulator output does not include values in the extremes of the true range for y_1. The extremes would then be poorly covered by the resulting design for the second emulator, meaning prediction from the chain at the values of the model 1 inputs $\tilde{\mathbf{x}}_1$ which lead to extreme values of y_1 may be unreliable. This could be overcome by simply adding some fixed amount to the range of y_1 when constructing its design space, but this risks placing design points at values of y_1 which may not in fact be in its true range.

Perhaps more significantly, sequential design for chains of models has the potential for significant improvement over the simple method presented in Section 4.3. This could include algorithms for sequential design which make better use of the available resources than can be achieved by simply applying standard methods for sequential design for a stand-alone emulator. By accounting for potential differences in computational cost and output variation between the individual models in the chain, it may be possible to learn significantly more about the behaviour of the chain without a substantial increase in computational cost. Learning more about the behaviour of the models allows better emulators to be built, so this could be extremely beneficial in terms of making predictions from the chain as a whole. Section 4.4 sets out some initial ideas as to how this could be achieved, but further research and implementation of these ideas is required.

9.3 Future work: sensitivity analysis

Sensitivity analysis is another area in which our work could be expanded substantially. While methods for the analysis of the final model are available, further work is required on analysis of the chain as a whole. As discussed in Section 5.6, the most plausible route to full sensitivity analysis for a chain under the assumptions required in Sections 3.4 and 3.6 is via a derivation for the covariance between two independent predictions from the linked emulator. A natural way to achieve this is by a similar process to the derivation of the posterior predictive variance of a single linked emulator output in Section 3.3. We present the beginnings of this below, but have been unable to complete the derivation.

Let Y_2 and Y_2' be two independent realisations of the linked emulator at distinct input points, and let $\tilde{\mathbf{x}}_1'$, $\tilde{\mathbf{x}}_2'$, y_1' and $\mathbf{c}_{n+1,2}'$ be the associated model 1 inputs, model 2 inputs, model 1 output and vector of correlations with the model 2 design points respectively. For ease of notation, let

$$Cov^*(Y_2, Y_2') = Cov[Y_2, Y_2' | \tilde{\mathbf{x}}_2, \tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2', \tilde{\mathbf{x}}_1', \mathbf{Y}_{n,2}, \mathbf{Y}_{n,1}]$$

be the covariance between the two realisations given all of the inputs except y_1 and y_1'. Also, let

$$c^*(Y_2, Y_2') = Cov[Y_2, Y_2' | \tilde{\mathbf{x}}_2, \tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2', \tilde{\mathbf{x}}_1', \mathbf{Y}_{n,2}, \mathbf{Y}_{n,1}, y_1, y_1']$$

be the covariance between the two realisations given all of the inputs including y_1 and y_1' (which are unknown beyond their probability distributions). The law of total covariance allows us to express $Cov^*(Y_2, Y_2')$ in terms of $c^*(Y_2, Y_2')$, μ_2 and μ_2' as

$$Cov^*(Y_2, Y_2') = Cov(\mu_2, \mu_2') + E[c^*(Y_2, Y_2')] \tag{9.1}$$

The second term of (9.1) is given by

$$E[c^*(Y_2, Y_2')] = E[\sigma_{z,2}^2 \{\mathbf{R}[(\tilde{\mathbf{x}}_2, y_1)^T, (\tilde{\mathbf{x}}_2', y_1')^T] - \mathbf{c}_{n+1,2}^T \mathbf{C}_2 \mathbf{c}_{n+1,2}'\}]$$

which reduces to

$$\begin{aligned} E[c^*(Y_2, Y_2')] =& \sigma_{z,2}^2 \prod_{d=1}^{q_2} \exp[-b_d(x_{2,d} - x_{2,d}')^2] E\Big\{ \exp[-b_{y1}(y_1 - y_1')^2] \Big\} \\ & - \sigma_{z,2}^2 E[\mathbf{c}_{n+1,2}^T \mathbf{C}_2 \mathbf{c}_{n+1,2}'] \,. \end{aligned} \tag{9.2}$$

Let

$$u = \mathbf{c}_{n+1,2}^T \mathbf{C}_2 \mathbf{c}'_{n+1,2}\,.$$

For the expectation in the second term of (9.2), we thus have

$$E[u] = E\Big[\sum_{i=1}^{k}\sum_{j=1}^{k} \exp\{-b_{y1}(\mathbf{x}_{2,i}^{(q_2+1)} - y_1)^2\} P_1 C_2^{(i,j)} \exp\{-b_{y1}(\mathbf{x}_{2,j}^{'(q_2+1)} - y'_1)^2\} P_2\Big]\,,$$

where

$$P_1 = \prod_{d=1}^{q_2} \exp\{-b_d(\mathbf{x}_{2,i}^{(d)} - \mathbf{x}_{2,n+1}^{(d)})^2\}$$

and

$$P_2 = \prod_{d=1}^{q_2} \exp\{-b_d(\mathbf{x}_{2,j}^{'(d)} - \mathbf{x}_{2,n+1}^{'(d)})^2\}\,.$$

Following a similar approach to that in Chapter 3 leads to

$$E[u] = \sum_{i=1}^{k}\sum_{j=1}^{k} C_2^{(i,j)} \prod_{d=1}^{p} \exp\{-b_d[(\mathbf{x}_{2,i}^{(d)} - \mathbf{x}_{2,n+1}^{(d)})^2 + (\mathbf{x}_{2,j}^{'(d)} - \mathbf{x}_{2,n+1}^{'(d)})^2]\} I^*_{i,j}$$

where

$$\begin{aligned} I^*_{i,j} = \int\int \frac{1}{2\pi\sigma_1\sigma'_1} \exp\Big\{-\frac{(y_1-\mu_1)^2}{2\sigma_1^2} - \frac{(y'_1-\mu'_1)^2}{2\sigma_1'^2}\Big\} \\ \exp\{-b_{y1}[(\mathbf{x}_{2,i}^{(q_2+1)} - y_1)^2 + (\mathbf{x}_{2,j}^{'(q_2+1)} - y'_1)^2]\} dy_1 dy'_1\,. \end{aligned} \qquad (9.3)$$

For ease of notation, let

$$W_1 = \exp\Bigg\{ - \frac{\mu_1^2 + 2\sigma_1^2 b_{y1}[\mathbf{x}_{2,i}^{(q_2+1)}]^2 - \frac{(\mu_1 + 2\sigma_1^2 b_{y1}\mathbf{x}_{2,i}^{(q_2+1)})^2}{1+2\sigma_1^2 b_{y1}}}{2\sigma_1^2}\Bigg\}\,,$$

and analogously,

$$W'_1 = \exp\Bigg\{ - \frac{\mu_1'^2 + 2\sigma_1'^2 b_{y1}[\mathbf{x}_{2,j}^{'(q_2+1)}]^2 - \frac{(\mu'_1 + 2\sigma_1'^2 b_{y1}\mathbf{x}_{2,j}^{'(q_2+1)})^2}{1+2\sigma_1'^2 b_{y1}}}{2\sigma_1'^2}\Bigg\}\,.$$

We also define

$$V_1 = \frac{(1+2\sigma_1^2 b_{y1})\Big[y_1 - \frac{\mu_1 + 2\sigma_1^2 b_{y1}\mathbf{x}_{2,i}^{(q_2+1)}}{1+2\sigma_1^2 b_{y1}}\Big]^2}{2\sigma_1^2}\,,$$

and

$$V_1' = \frac{(1+2\sigma_1'^2 b_{y1})\left[y_1' - \frac{\mu_1' + 2\sigma_1'^2 b_{y1}\mathbf{x}_{2,j}'^{(q_2+1)}}{1+2\sigma_1'^2 b_{y1}}\right]^2}{2\sigma_1'^2}.$$

Further algebraic manipulation of (9.3) leads to

$$I_{i,j}^* = C_1 I_1$$

where

$$C_1 = \frac{1}{\sqrt{(1+2\sigma_1^2 b_{y1})(1+2\sigma_1'^2 b_{y1})}} W_1 W_1'$$

and

$$I_1 = \int\int \frac{\sqrt{1+2\sigma_1^2 b_{y1}}\sqrt{1+2\sigma_1'^2 b_{y1}}}{2\pi\sigma_1\sigma_1'} \exp\{-V_1 - V_1'\} dy_1 dy_1'$$

The integrand in I_1 is the density of a bivariate normal distribution with zero correlation, where the other parameters are given by

$$\mu(y_1) = \frac{\mu_1 + 2\sigma_1^2 b_{y1}\mathbf{x}_{2,i}^{(q_2+1)}}{1+2\sigma_1^2 b_{y1}};$$

$$\mu(y_1') = \frac{\mu_1' + 2\sigma_1'^2 b_{y1}\mathbf{x}_{2,j}'^{(q_2+1)}}{1+2\sigma_1'^2 b_{y1}};$$

$$\sigma(y_1) = \frac{\sigma_1}{\sqrt{1+2\sigma_1^2 b_{y1}}};$$

$$\sigma(y_1') = \frac{\sigma_1'}{\sqrt{1+2\sigma_1'^2 b_{y1}}}.$$

When integrated with respect to y_1 and y_1', this is equal to 1, so $I_1 = 1$, leaving

$$I_{i,j}^* = C_1,$$

but the complicated form of C_1, W_1 and W_1' suggests that further simplification is required before this quantity is used in the full covariance expression.

The first term of (9.1) is

$$Cov(\mu_2, \mu_2') = E(\mu_2\mu_2') - E(\mu_2)E(\mu_2').$$

$E(\mu_2)$ and $E(\mu_2')$ are found from equation (3.7), so we only need to calculate

$$E(\mu_2\mu_2') = E_{y1}\{[\beta_{2,0} + \mathbf{c}_{n+1,2}^T \mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \beta_{2,0})][\beta_{2,0} + \mathbf{c}_{n+1,2}'^T \mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \beta_{2,0})]\},$$

which simplifies to

$$\begin{aligned}E(\mu_2\mu_2') =&\beta_{2,0}^2 + \beta_{2,0}E[\mathbf{c}_{n+1,2}^T\mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \beta_{2,0})] + \beta_{2,0}E[\mathbf{c}_{n+1,2}^{\prime T}\mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \beta_{2,0})] \\ &+ E[\{\mathbf{c}_{n+1,2}^T\mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \beta_{2,0})\}\{\mathbf{c}_{n+1,2}^{\prime T}\mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \beta_{2,0})\}]\end{aligned} \tag{9.4}$$

An expression for $E[\mathbf{c}_{n+1,2}^T\mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \beta_{2,0})]$ (and by extension the case where $\mathbf{c}_{n+1,2}^{\prime T}$ is used instead) was derived in (3.9):

$$E[\mathbf{c}_{n+1,2}^T\mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \beta_{2,0})] = \sum_{i=1}^{k} a^{(i)} \prod_{j=1}^{q_2} \exp\{-b_j(\mathbf{x}_{2,i}^{(j)} - \mathbf{x}_{2,n+1}^{(j)})^2\} I_i \, ,$$

where I_i is given in (3.11). To deal with the remaining expectation, note that

$$\begin{aligned}\{\mathbf{c}_{n+1,2}^T\mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \beta_{2,0})\}\{\mathbf{c}_{n+1,2}^{\prime T}\mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \beta_{2,0})\} &= (\mathbf{c}_{n+1,2}^T\,\mathbf{a})(\mathbf{c}_{n+1,2}^{\prime T}\,\mathbf{a}) \\ &= \sum_{i=1}^{k}\sum_{j=1}^{k} c_2^{(i)} c_2^{\prime(j)} a^{(i)} a^{(j)} \, ,\end{aligned}$$

so

$$E\Big[\{\mathbf{c}_{n+1,2}^T\mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \beta_{2,0})\}\{\mathbf{c}_{n+1,2}^{\prime T}\mathbf{C}_2^{-1}(\mathbf{y}_{n,2} - \beta_{2,0})\}\Big] = E\Bigg[\sum_{i=1}^{k}\sum_{j=1}^{k} c_2^{(i)} c_2^{\prime(j)} a^{(i)} a^{(j)}\Bigg] .$$

Algebraic manipulation similar to that seen in Chapter 3 gives

$$E\Bigg[\sum_{i=1}^{k}\sum_{j=1}^{k} c_2^{(i)} c_2^{\prime(j)} a^{(i)} a^{(j)}\Bigg] = \sum_{i=1}^{k}\sum_{j=1}^{k} a^{(i)} a^{(j)} P_3 I_{i,j}^* \, ,$$

where

$$P_3 = \prod_{d=1}^{q_2} \exp\{-b_d[(\mathbf{x}_{2,i}^{(d)} - \mathbf{x}_{2,n+1}^{(d)})^2 + (\mathbf{x}_{2,j}^{(d)} - \mathbf{x}_{2,n+1}^{\prime(d)})^2]\} \, .$$

As above, this expression depends on further simplification of $I_{i,j}^*$ but is otherwise complete.

For ease of notation, let

$$c_y^* = \exp[-b_{y1}(y_1 - y_1')^2] \, .$$

The expectation in the first term of (9.2) is then

$$E(c_y^*) = \int\int f(y_1) f(y_1') \exp[-b_{y1}(y_1 - y_1')^2] dy_1 dy_1' \, ,$$

which, when the distributions are substituted in and like terms collected, reduces to

$$E(c_y^*) = \frac{1}{2\pi\sigma_1\sigma_1'}\int\int \exp\left[-\frac{N_1}{2\sigma_1^2\sigma_1'^2}\right]dy_1dy_1'\,,$$

where

$$N_1 = \sigma_1'^2(1+2\sigma_1^2 b_{y1})y_1^2 - 2\sigma_1'^2\mu_1 y_1 - 2\sigma_1^2 y_1'\mu_1' - \sigma_1^2(1+2\sigma_1'^2 b_{y1})y_1'^2 - 4\sigma_1^2\sigma_1'^2 b_{y1}y_1y_1'\,.$$

However we have been unable to make further progress on this integral. This is the main reason why the covariance derivation is incomplete.

Even if this were completed, it would provide full sensitivity analysis only when the conditions required for the theoretical approximation to hold are met. Sensitivity analysis for chains with different correlation or regression structures, or where the process variances are integrated out of the individual emulators, would require a different method.

One option is to consider a form of decomposition of the Sobol' indices of a chain of models. For example, consider a two-model chain with final output y_2, where the second model includes the input y_1 which is the output of an earlier model. Let

$$S_{y2}(x_{1,1}) = \frac{var\{E(Y_2|X_{1,1}=x_{1,1})\}}{var(Y_2)}\,. \tag{9.5}$$

be the Sobol' index for the input $x_{1,1}$ with respect to the output y_2. We have

$$S_{y2}(y_1) = \frac{var\{E(Y_2|Y_1=y_1)\}}{var(Y_2)}$$

and

$$S_{y1}(x_{1,1}) = \frac{var\{E(Y_1|X_{1,1}=x_{1,1})\}}{var(Y_1)}\,.$$

A simple approximation could be provided by

$$S_{y2}(x_{1,1}) \approx S_{y2}(y_1)\times S_{y1}(x_{1,1})\,.$$

The logic for this is as follows: $S_{y2}(y_1)$ corresponds to the proportion of the variance in y_2 that is explained by y_1. Similarly, $S_{y1}(x_{1,1})$ is the proportion of the variance in y_1 explained by $x_{1,1}$. It therefore appears natural that the product of the two indices should correspond to the proportion of the variance in y_2 which is explained by $x_{1,1}$, since there is no additional source of uncertainty in y_1 other than its inputs. However, no theoretical result exists to demonstrate that this is a good approximation for two (or more) complex non-linear simulators, although initial simulation studies to test this empirically for simple chains were encouraging.

Another possible solution is to bypass the chain of emulators by using a single approximation. We have already seen that a composite emulator does not perform well for the chains of computational models covered in this thesis. However, for sensitivity analysis a composite emulator has two major advantages: it is significantly less computationally intensive to make predictions from than a linked emulator, and allows the pre-existing sensitivity analysis results from Section 5.3 to be used without alteration.

A possible compromise, therefore, is to build a linked emulator as discussed before, but then approximate this again by a single emulator. While this adds another layer of uncertainty to the process, it neatly bypasses many of the problems associated with both the linked and the composite emulators when conducting sensitivity analysis. Since the linked emulator is generally much less computationally intensive than the chain of simulators (but still too intensive for sensitivity analysis to be done directly), a relatively large set of design points could reasonably be used. The linked emulator is however a stochastic function, so any emulator to it must account for this; the methods discussed in Section 2.4 would need to be used. The design points used to train the linked emulator could themselves be used as design points for the single stochastic emulator, with a known uncertainty in the linked emulator output of 0 at these points. This somewhat esoteric approach has not been implemented, but may be of interest in the future.

9.4 Future work: other areas

There are also other ways in which our methodology could be extended. The chains of models considered in this thesis all obey a relatively restrictive set of assumption, and a significantly wider class of problems could also be approached using similar methods. One natural extension would be to unify our framework with that of Kyzyurova et al. (2018), in which multiple inputs to the same computational model may come from other models. This would allow more complicated structures than just a chain to be considered. Such a unification process should be relatively straightforward, as adding additional inputs from other models does not change the theoretical results for the linked emulator substantially, so adapting the theoretical results presented here to the more general framework may not present a great challenge.

The individual emulators considered here are also somewhat restrictive. Incorporating Markov chain Monte Carlo inference for the correlation parameters of the Gaussian process emulators would allow the uncertainty in these parameters to be incorporated into our predictions. Emulators which account for non-stationarity in the underlying computational models may also have a role to play in broadening the set of problems which our methodology can be applied to.

The framework we present for emulation applies to a specific type of model, with a single deterministic output and continuous inputs. Models with multiple outputs and/or categorical inputs could also be considered as part of the chain. Models with stochastic instead of deterministic output are another avenue worthy of exploration. These extensions would require incorporating some of the processes introduced in Section 2.4. Given that these processes change the underlying nature of the emulators involved, it is likely that the theoretical results discussed in this thesis would no longer hold, so new theoretical derivations would be required if anything beyond a simple Monte Carlo method were to be used.

Finally, the application to a chain of models for probability of casualty from a chain of models for a CBR release could also be expanded. Real world dispersion and casualty models can be far more complex than those considered here. There are many more inputs that could be taken into account in addition to release rate, release duration and release time. Examples include the release location and meteorological conditions such as wind speed and wind direction, the latter of which are in fact inputs to the single model for a CBR release in Chapter 5. It would also be useful to consider the probability of casualty across a range of locations, instead of at a single point; this would need the emulator for the final model to be able to handle multiple outputs, as discussed above. A full study of the CBR problem would include at least one additional model in the chain, for the meteorological conditions, and potentially another for the placement of sensors to detect a release.

Bibliography

R.J. Adler. *The Geometry of Random Fields.* John Wiley, 1981.

I. Andrianakis and P. G. Challenor. The effect of the nugget on Gaussian process emulators of computer models. *Computational Statistics and Data Analysis*, 56: 4215–4228, 2012.

S. Ba. Package ‘slhd’. R package vignette, 2015.

S. Banerjee. Bayesian linear model: Gory details. http://www.biostat.umn.edu/~{}ph7440/pubh7440/BayesianLinearModelGoryDetails.pdf, 2010.

L. S. Bastos and A. O’Hagan. Diagnostics for Gaussian process emulators. *Technometrics*, 51:425–438, 2009.

J. Beck and S. Guillas. Sequential design with mutual information for computer experiments (MICE): Emulation of a tsunami model. *Journal of Uncertainty Quantification*, 4:739–766, 2016.

J. Berkson. Application of the logistic function to bio-assay. *Journal of the American Statistical Association*, 39:357–365, 1944.

I. Billonis and N. Zabaras. Multi-output local Gaussian process regression: Applications to uncertainty quantification. *Journal of Computational Physics*, 231(17):5718–5746, 2012.

M. Binois, J. Huang, R. B. Gramacy, and M. Ludkovski. Replication or exploration? Sequential design for stochastic simulation experiments. *Technometrics*, pages 1–43, 2018.

C.I. Bliss. The method of probits. *Science*, 79(2037):38–39, 1934.

V.E. Bowman and D.C. Woods. Emulation of multivariate simulators using thin plate splines with application to atmospheric dispersion. *SIAM/ASA Journal on Uncertainty Quantification*, 4:1323–1344, 2016.

F. Campolongo, J. Cariboni, and A. Saltelli. An effective screening design for sensitivity analysis of large models. *Environmental Modelling and Software*, 22:1509–1518, 2007.

J.Q. Candela, A. Girard, J. Larsen, and C.E. Rasmussen. Propagation of uncertainty in Bayesian kernel models - application to multiple-step ahead forecasting. *Proceedings* *of the 2003 IEEE Conference on Acoustics, Speech, and Signal Processing*, 2:II–701, 2003.

W. F. Caselton and J. V. Zidek. Optimal monitoring network designs. *Statistics & Probability Letters*, 2(4):223–227, 1984.

D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active learning with statistical models. *Journal of Artificial Intelligence Research*, 4:129–145, 1996.

S. Conti and A. O'Hagan. Bayesian emulation of complex multi-output and dynamic computer models. *Journal of Statistical Planning and Inference*, 140(3):640–651, 2010.

A. C. Damianou and N. D. Lawrence. Deep Gaussian Processes. In *Proceedings of the 16th International Conference on Artificial Intelligence and Statistics*, 2013.

A. C. Damianou and N. D. Lawrence. Uncertainty propagation in Gaussian process pipelines. In *NIPS workshop on modern non-parametrics*, 2014.

G.M. Dancik and K.S. Dorman. mlegp: statistical analysis for computer models of biological systems using R. *Bioinformatics*, 2008.

S. Duane, A.D. Kennedy, B. Pendleton, and D. Roweth. Hybrid Monte Carlo. *Physics Letters B*, 195:216–222, 1987.

D. Eddelbuettel. *Seamless R and C++ Integration with Rcpp*. Springer, 2013.

D. Eddelbuettel and C. Sanderson. RcppArmadillo: Accelerating R with high-performance C++ linear algebra. *Computational Statistics and Data Analysis*, 71, 2014.

J. Fan. Comment on "Wavelets in statistics: A review" by A. Antoniadis. *Journal of the Italian Statistical Society*, 6(2):131–138, 1997.

K. Fang, R. Li, and A. Sudjianto. *Design and Modelling for Computer Experiments*. Chapman and Hall, Boca Raton, 2006.

D. Gamerman and H.F. Lopes. *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference (Second Edition)*. Chapman and Hall, 2006.

A. Girard, C. E. Rasmussen, J.Q. Candela, and R. Murray-Smith. Gaussian process priors with uncertain inputs - application to multiple-step ahead time series forecasting. In *Advances in Neural Information Processing Systems*, volume 15, pages 529–536, 2002.

P. W. Goldberg, C. K. I. Williams, and C.M. Bishop. Regression with input-dependent noise: A Gaussian process treatment. In *Advances in Neural Information Processing Systems*, volume 10, pages 493–499, 1998.

R. B. Gramacy and H. K. H. Lee. Bayesian treed Gaussian process models with an application to computer modeling. *Journal of the American Statistical Association*, 103:1119–1130, 2008.

R. B. Gramacy and H. K. H. Lee. Adaptive design and analysis of supercomputer experiments. *Technometrics*, 51(2):130–145, 2009.

R. B. Gramacy and H. K. H. Lee. Cases for the nugget in modeling computer experiments. *Statistics and Computing*, 22(3):713–722, 2012.

R. B. Gramacy and M. A. Taddy. Categorical inputs, sensitivity analysis, optimization and importance tempering with tgp version 2, an r package for treed Gaussian process models. *Journal of Statistical Software*, 33(6), 2010.

M. Gu, X. Wang, and J.O. Berger. Robust Gaussian stochastic process emulation. *The Annals of Statistics*, 46(6A):3038–3066, 2018.

W.K. Hastings. Monte Carlo sampling methods using Markov Chains and their applications. *Biometrika*, 57(1):97–109, 1970.

R.G. Haylock and A. O'Hagan. On inference for outputs of computationally expensive algorithms with uncertainty on the inputs. *Bayesian Statistics*, 5:629–637, 1996.

D. Higdon, J. Gattiker, B. Williams, and M. Rightley. Computer model calibration using high dimensional output. *Journal of the American Statistical Association*, 103 (482):570–583, 2008.

R. L. Iman and W. J. Conover. A distribution-free approach to inducing rank correlation among input variates. *Communication in Statistics - Simulation and Computation*, 11(3):311–334, 1982.

H. Jeffreys. *Theory of Probability.* Oxford University Press, 1961.

M.E. Johnson, L.M. Moore, and D. Ylvisaker. Minimax and maximin distance designs. *Journal of Statistical Planning and Inference*, 26:131–148, 1990.

V. R. Joseph, L. Gu, and W. Myers. Space-filling designs for robustness experiments. *Technometrics*, 61(1):24–37, 2019.

J. Jun and I. Horace. Active learning with SVM. In J. Ramón, R. Dopico, J. Dorado, and A. Pazos, editors, *Encyclopedia of Artificial Intelligence*, volume 3, pages 1–7. ICI Global, 2009.

K. Kersting, C. Plagemann, P. Pfaff, and W. Burgard. Most likely heteroscedastic Gaussian process regression. In *Proceedings of the International Conference on Machine Learning*, pages 393–400, 2007.

A. Konyukhov, P. Vielsack, and K. Schweizerhof. On coupled models of anisotropic contact surfaces and their experimental validation. *Wear*, 264(7-8):579–588, 2008.

A. Krause, A. Singh, and C. Guestrin. Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9:235–284, 2008.

K.N. Kyzyurova, J.O. Berger, and R.L. Wolpert. Coupling computer models through linking their statistical emulators. *SIAM/ASA Journal on Uncertainty Quantification (JUQ)*, 6(3):1151–1171, 2018.

L. Le Gratiet, C. Cannamela, and B. Iooss. A Bayesian approach for global sensitivity analysis of (multi-fidelity) computer codes. *SIAM/ASA Journal on Uncertainty Quantification (JUQ)*, 2:336–363, 2014.

R. Li and A. Sudjianto. Analysis of computer experiments using penalized likelihood in Gaussian kriging models. *Technometrics*, 47(2):111–120, 2005.

H. Lin and S. Yim. Coupled surge-heave motions of a moored system. I: Model calibration and parametric study. *Journal of Engineering Mechanics*, 132(6):671–680, 2006.

D. V. Lindley. On a measure of the information provided by an experiment. *Annals of Mathematical Statistics*, 27(4):986–1005, 1956.

D. MacKay. Information-based objective functions for active data selection. *Neural Computation*, 4(4):590–604, 1992.

S. Mak and V. R. Joseph. Minimax and minimax projection designs using clustering. *Journal of Computational and Graphical Statistics*, 27(1):166–178, 2018.

B. Matern. Spatial variation. *Meddelanden Fran Statens Skogs-Forskningsinstitut*, 49 (5), 1960.

M.D. McKay, R.J. Beckman, and W.J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21:239–245, 1979.

N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21 (6):1087–1092, 1953.

D. M. Morris. Factorial sampling plans for preliminary computational experiments. *Technometrics*, 33:161–174, 1991.

R. M. Neal. *Bayesian learning for neural networks*. Springer, 1996.

R. M. Neal. *Handbook of Markov Chain Monte Carlo*, chapter MCMC Using Hamiltonian Dynamics. Chapman and Hall, 2011.

F. Novometsky and S. Nadarajah. Package 'truncdist'. R package vignette, 2016.

E. Nummelin. *General Irreducible Markov Chains and Non-Negative Operators.* Cambridge University Press, 1984.

D. Nychka, Q. Yang, and J.A. Royle. *Constructing spatial designs for monitoring air pollution using subset regression.* Statistics for the Environment 3: Pollution Assessment and Control. Wiley, 1997.

D. Nychka, R. Furrer, J. Paige, and S. Sain. Package 'fields'. R package vignette, 2016.

J.E. Oakley and A. O'Hagan. Probabilistic sensitivity analysis of complex models: a Bayesian approach. *Journal of the Royal Statistical Society*, 66:751–769, 2004.

A. O'Hagan. Bayesian analysis of computer code outputs: A tutorial. *Reliability Engineering and System Safety*, 91:1290–1300, 2006.

H.D. Patterson and R. Thompson. Recovery of inter-block information when block sizes are unequal. *Biometrika*, 58(3):545–554, 1971.

A. Plumb. Metamodelling for hazard prediction. Master's thesis, University of Southampton, 2008.

R.L. Prentice. A generalization of the probit and logit methods for dose response curves. *Biometrics*, 32(4):761–768, 1976.

W.H. Press and G.R. Farrar. Recursive stratified sampling for multidimensional Monte Carlo integration. *Computers in Physics*, 4(190), 1990.

G. Pujol, B. Iooss, and A. Janon. Package 'sensitivity'. R package vignette, February 2017.

Z. G. Qian, H. Wu, and C. F. J. Wu. Gaussian process models for computer experiments with qualitative and quantitative factors. *Technometrics*, 50:383–396, 2009.

C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning.* MIT Press, Cambridge, MA, 2006.

G.O. Roberts, A. Gelman, and W.R. Gilks. Weak convergence and optimal scaling of random walk Metropolis algorithms. *The Annals of Applied Probability*, 7(1): 110–120, 1997.

O. Roustant, D. Ginsbourger, and Y. Deville. DiceKriging, DiceOptim: Two R Packages for the Analysis of Computer Experiments by Kriging-Based Metamodeling and Optimization. *Journal of Statistical Software*, 51(1), 2012.

J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn. Design and analysis of computer experiments (with discussion). *Statistical Science*, 4:409–435, 1989.

A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, and S. Tarantola. *Global Sensitivity Analysis.* Wiley, 2008.

T. J. Santner, B. J. Williams, and W. I. Notz. *The Design and Analysis of Computer Experiments*. Springer, New York, 2003.

M. Schonlau and W. J. Welch. *Screening Methods for Experimentation in Industry, Drug Discovery and Genetics*, chapter Screening the Input Variables to a Computer Model Via Analysis of Variance and Visualization, pages 308–327. Springer, 2006.

S. Seo, M. Wallat, T. Graepel, and K. Obermayer. Gaussian process regression: Active data selection and test point rejection. In *Proceedings of the International Joint Conference on Neural Networks*, volume 3, pages 241–246. IEEE, 2000.

A. F. Stein, R. R. Draxler, G. D. Rolph, B. J. B. Stunder, M. D. Cohen, and F. Ngan. NOAA's HYSPLIT Atmospheric Transport and Dispersion Modeling System. *Bulletin of the American Meteorological Society*, 2015.

M. L. Stein. *Interpolation of Spatial Data: Some Theory for Kriging.* Springer, 1999.

G. Stevens and S. Atamturktur. Mitigating error and uncertainty in partitioned analysis: A review of verification, calibration and validation methods for coupled simulations. *Archives of Computational Methods in Engineering*, pages 1–15, 2016.

M. Titsias and N. D. Lawrence. Bayesian Gaussian process latent variable model. *Journal of Machine Learning Research - Proceedings Track*, 9:844–851, 2010.

W. J. Welch, R. J. Buck, J. Sacks, H. P. Wynn, T. J. Mitchell, and M. D. Morris. Screening, predicting, and computer experiments. *Technometrics*, 1992.

Index